e-Chapter 6

Similarity-Based Methods



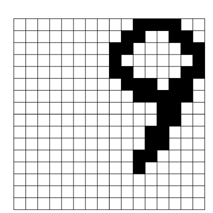
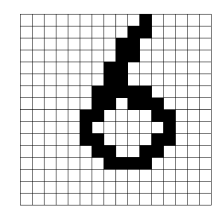
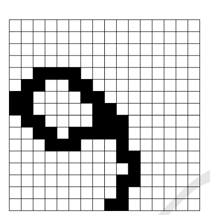
"It's a *manohorse*", exclaimed the confident little 5 year old boy. We call it the Centaur out of habit, but who can fault the kid's intuition? The 5 year old has never seen this thing before now, yet he came up with a reasonable classification for the beast. He is using the simplest method of learning that we know of – *similarity* – and yet it's effective: the child searches through his history for similar objects (in this case a man and a horse) and builds a classification based on these similar objects.

The method is simple and intuitive, yet when we get into the details, several issues need to be addressed in order to arrive at a technique that is quantitative and fit for a computer. The goal of this chapter is to build exactly such a quantitative framework for similarity based learning.

6.1 Similarity

The 'manohorse' is interesting because it requires a deep understanding of similarity: first, to say that the Centaur is similar to both man and horse; and, second, to decide that there is enough similarity to both objects so that neither can be excluded, warranting a new class. A good measure of similarity allows us to not only classify objects using similar objects, but also detect the arrival of a new class of objects (novelty detection).

A simple classification rule is to give a new input the class of the most similar input in your data. This is the 'nearest neighbor' rule. To implement the nearest neighbor rule, we need to first quantify the similarity between two objects. There are different ways to measure similarity, or equivalently dissimilarity. Consider the following example with 3 digits.

The two 9s should be regarded as very similar. Yet, if we naively measure similarity by the number of black pixels in common, the two 9s have only two in common. On the other hand, the 6 has many more black pixels in common with either 9, even though the 6 should be regarded as dissimilar to both 9s. Before measuring the similarity, one should preprocess the inputs, for example by centering, axis aligning and normalizing the size in the case of an image. One can go further and extract the relevant *features* of the data, for example size (number of black pixels) and symmetry as was done in Chapter 3. These practical considerations regarding the nature of the learning task, though important, are not our primary focus here. We will assume that through domain expertise or otherwise, features have been constructed to identify the important dimensions, and if two inputs differ in these dimensions, then the inputs are likely to be dissimilar. Given this assumption, there are well established ways to measure similarity (or dissimilarity) in different contexts.

6.1.1 Similarity Measures

For inputs $\mathbf{x}, \mathbf{x}'$ which are vectors in $\mathbb{R}^d$, we can measure dissimilarity using the standard Euclidean distance,

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|.$$

The smaller the distance, the more similar are the objects corresponding to inputs $\mathbf{x}$ and $\mathbf{x}'$.[1] For Boolean features the Euclidean distance is the square root of the well known *Hamming distance*. The Euclidean distance is a special case of a more general distance measure which can be defined for an arbitrary positive semi-definite matrix Q:

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^{\mathrm{T}} \mathrm{Q} (\mathbf{x} - \mathbf{x}').$$

A useful special case, known as the *Mahalanobis distance* is to set $\mathrm{Q} = \Sigma^{-1}$, where Σ is the covariance matrix [2] of the data. The Mahalanobis distance metric depends on the data set. The main advantage of the Mahalanobis distance over the standard Euclidean distance is that it takes into account correlations among the data dimensions and scale. A similar effect can be

[1] An axiomatic treatment of similarity that goes beyond metric-based similarity appeared in "Features of Similarity", A. Tversky, *Psychological Review*, 84(4), pp 327–352, 1977.

[2] $\Sigma = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} - \bar{\mathbf{x}}\bar{\mathbf{x}}^{\mathrm{T}}$, where $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$.

accomplished by first using input preprocessing to standardize the data (see Chapter 9) and then using the standard Euclidean distance. Another useful measure, especially for Boolean vectors, is the *cosine similarity*,

$$\text{CosSim}(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x} \bullet \mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}.$$

The cosine similarity is the cosine of the angle between the two vectors, CosSim $\in [-1, 1]$, and larger values indicate greater similarity. When the objects represent sets, then the set similarity or *Jaccard coefficient* is often used. For example, consider two movies which have been watched by two different sets of users S_1, S_2. We may measure how similar these movies are by how similar the two sets S_1 and S_2 are:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|};$$

$1 - J(S_1, S_2)$ can be used as a measure of distance which conveniently has the properties that a metric formally satisfies, such as the triangle inequality. We focus on the Euclidean distance which is also a metric; many of the algorithms, however, apply to arbitrary similarity measures.

Exercise 6.1

(a) Give two vectors with very high cosine similarity but very low Euclidean distance similarity. Similarly, give two vectors with very low cosine similarity but very high Euclidean distance similarity.

(b) If the origin of the coordinate system changes, which measure of similarity changes? How will this affect your choice of features?

6.2 Nearest Neighbor

Simple rules survive; and, the nearest neighbor technique is perhaps the simplest of all. We will summarize the entire algorithm in a short paragraph. But, before we forge ahead, let's not forget the two basic competing principles laid out in the first five chapters: any learning technique should be expressive enough that we can fit the data and obtain low E_{in}; however, it should be reliable enough that a low E_{in} implies a low E_{out}.

The *nearest neighbor rule* is embarrassingly simple. There is no training phase (or no 'learning' so to speak). The entire algorithm is specified by how one computes the final hypothesis $g(\mathbf{x})$ on a test input $\mathbf{x}$. Recall that the data set is $\mathcal{D} = (\mathbf{x}_1, y_1) \ldots (\mathbf{x}_N, y_N)$, where $y_n = \pm 1$. To classify the test point $\mathbf{x}$, find the nearest point to $\mathbf{x}$ in the data set (the nearest neighbor), and use the classification of this nearest neighbor.

Formally speaking, reorder the data according to distance from $\mathbf{x}$ (breaking ties using the data point's index for simplicity). We write $(\mathbf{x}_{[n]}(\mathbf{x}), y_{[n]}(\mathbf{x}))$

for the nth such reordered data point with respect to $\mathbf{x}$. We will drop the dependence on $\mathbf{x}$ and simply write $(\mathbf{x}_{[n]}, y_{[n]})$ when the context is clear. So,

$$d(\mathbf{x}, \mathbf{x}_{[1]}) \leq d(\mathbf{x}, \mathbf{x}_{[2]}) \leq \cdots \leq d(\mathbf{x}, \mathbf{x}_{[N]})$$

The final hypothesis is

$$g(\mathbf{x}) = y_{[1]}(\mathbf{x})$$

($\mathbf{x}$ is classified by just looking at the label of the nearest data point to $\mathbf{x}$). This simple nearest neighbor rule admits a nice geometric interpretation, shown for two dimensions in Figure 6.1.

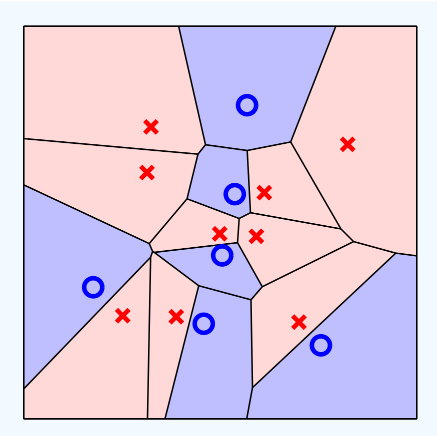


Figure 6.1: Nearest neighbor Voronoi tessellation.

The shading illustrates the final hypothesis $g(\mathbf{x})$. Each data point $\mathbf{x}_n$ 'owns' a region defined by the points closer to $\mathbf{x}_n$ than to any other data point. These regions are convex polytopes (convex regions whose faces are hyperplanes), some of which are unbounded; in two dimensions, we get convex polygons. The resulting set of regions defined by such a set of points is called a *Voronoi (or Dirichlet) tessellation* of the space. The final hypothesis g is a Voronoi tessellation with each region inheriting the class of the data point that owns the region. Figure 6.2(a) further illustrates the nearest neighbor classifier for a sample of 500 data points from the digits data described in Chapter 3.

The clear advantage of the nearest neighbor rule is that it is simple and intuitive, easy to implement and there is no training. It is expressive, as it achieves zero in-sample error (as can immediately be deduced from Figure 6.1), and as we will soon see, it is reliable. A practically important cosmetic upside, when presenting to a client for example, is that the classification of a test object is easy to 'explain': just present the similar object on which the classification is based. The main disadvantage is the computational overhead.

VC dimension. The nearest neighbor rule fits within our standard supervised learning framework with a very large hypothesis set. Any set of n labeled points induces a Voronoi tessellation with each Voronoi region assigned to a class; thus any set of n labeled points defines a hypothesis. Let $\mathcal{H}_n$ be the hypothesis set containing all hypotheses which result from some labeled Voronoi tessellation on n points. Let $\mathcal{H} = \cup_{n=1}^{\infty} \mathcal{H}_n$ be the union of all these hypothesis sets; this is the hypothesis set for the nearest neighbor rule. The learning algorithm picks the particular hypothesis from $\mathcal{H}$ which corresponds to the realized labeled data set; this hypothesis is in $\mathcal{H}_N \subset \mathcal{H}$. Since the training error is zero no matter what the size of the data set, the nearest neighbor rule is non-falsifiable [3] and the VC-dimension of this model is infinite. So, from the VC-worst-case analysis, this spells doom. A finite VC-dimension would have been great, and it would have given us one form of reliability, namely that E_{out} is close to E_{in} and so minimizing E_{in} works. The nearest neighbor method is reliable in another sense, and we are going to need some new tools if we are to argue the case.

6.2.1 Nearest Neighbor is 2-Optimal

Using a probabilistic argument, we will show that, under reasonable assumptions, the nearest neighbor rule has an out-of-sample error that is at most twice the *minimum possible out-of-sample error*. The success of the nearest neighbor algorithm relies on the nearby point $\mathbf{x}_{[1]}(\mathbf{x})$ having the same classification as the test point $\mathbf{x}$. This means two things: there is a 'nearby' point in the data set; and, the target function is reasonably smooth so that the classification of this nearest neighbor is indicative of the classification of the test point.

We model the target value, which is ± 1, as noisy and define

$$\pi(\mathbf{x}) = \mathbb{P}[y = +1 | \mathbf{x}].$$

A data pair $(\mathbf{x}, y)$ is obtained by first generating $\mathbf{x}$ from the input probability distribution $P(\mathbf{x})$, and then y from the conditional distribution $\pi(\mathbf{x})$. We can relate $\pi(\mathbf{x})$ to a deterministic target function $f(\mathbf{x})$ by observing that if $\pi(\mathbf{x}) \geq \frac{1}{2}$, then the optimal prediction is $f(\mathbf{x}) = +1$; and, if $\pi(\mathbf{x}) < \frac{1}{2}$ then the optimal prediction is $f(\mathbf{x}) = -1$.[4]

[3] Recall that a hypothesis set is non-falsifiable if it can fit any data set.
[4] When $\pi(\mathbf{x}) = \frac{1}{2}$, we break the tie in favor of $+1$.

Exercise 6.2

Let

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \pi(\mathbf{x}) \geq \frac{1}{2}, \\ -1 & \text{otherwise.} \end{cases}$$

Show that the probability of error on a test point $\mathbf{x}$ is

$$e(f(\mathbf{x})) = \mathbb{P}[f(\mathbf{x}) \neq y] = \min\{\pi(\mathbf{x}), 1 - \pi(\mathbf{x})\}$$

and $e(f(\mathbf{x})) \leq e(h(\mathbf{x}))$ for any other hypothesis h (deterministic or not).

The error $e(f(\mathbf{x}))$ is the probability that $y \neq f(\mathbf{x})$ on the test point $\mathbf{x}$. The previous exercise shows that $e(f(\mathbf{x})) = \min\{\pi(\mathbf{x}), 1 - \pi(\mathbf{x})\}$, which is the minimum probability of error possible on test point $\mathbf{x}$. Thus, the best possible out-of-sample misclassification error is the expected value of $e(f(\mathbf{x}))$,

$$E^*_{\text{out}} = \mathbb{E}_{\mathbf{x}}[e(f(\mathbf{x}))] = \int d\mathbf{x} \;\; P(\mathbf{x}) \min\{\pi(\mathbf{x}), 1 - \pi(\mathbf{x})\}.$$

If we are to infer $f(\mathbf{x})$ from a nearest neighbor using $f(\mathbf{x}_{[1]})$, then $\pi(\mathbf{x})$ should not be varying too rapidly and $\mathbf{x}_{[1]}$ should be close to $\mathbf{x}$. We require that $\pi(\mathbf{x})$ should be continuous.[5] To ensure that there is a near enough neighbor $\mathbf{x}_{[1]}$ for *any* test point $\mathbf{x}$, it will suffice that N be large enough. We now show that, if $\pi(\mathbf{x})$ is continuous and N is large enough, then the simple nearest neighbor rule is reliable; specifically, it achieves an out-of-sample error which is at most twice the minimum achievable error (hence the title of this section).

Let $\{\mathcal{D}_N\}$ be a sequence of data sets with sizes $N = 1, 2, \ldots$ generated according to $P(\mathbf{x}, y)$ and let $\{g_N\}$ be the associated nearest neighbor decision rules for each data set. For N large enough, we will show that

$$E_{\text{out}}(g_N) \leq 2E^*_{\text{out}}.$$

So, even though the nearest neighbor rule is non-falsifiable, the test error is at most a factor of 2 worse than optimal (in the asymptotic limit, i.e., for large N). The nearest neighbor rule is reliable; however, E_{in}, which is always zero, will never match E_{out}. We will never have a good theoretical estimate of E_{out} (so we will not know our final performance), but we know that you cannot do much better. The formal statement we can make is:

Theorem 6.1. For any $\delta > 0$, and any continuous noisy target $\pi(\mathbf{x})$, there is a large enough N for which, with probability at least $1 - \delta$, $E_{\text{out}}(g_N) \leq 2E^*_{\text{out}}$.

The intuition behind the factor of 2 is that $f(\mathbf{x})$ makes a mistake only if there is an error on the test point $\mathbf{x}$, whereas the nearest neighbor classifier makes a mistake if there is an error on the test point *or* on the nearest neighbor (which adds up to the factor of 2).

[5]Formally, it is only required that the assumptions hold on a set of probability 1.

More formally, consider a test point $\mathbf{x}$. The nearest neighbor classifier makes an error on $\mathbf{x}$ if $y = 1$ and $y_{[1]} = -1$ or if $y = -1$ and $y_{[1]} = 1$.

$$\begin{aligned}\mathbb{P}[g_N(\mathbf{x}) \neq y] &= \pi(\mathbf{x}) \cdot (1 - \pi(\mathbf{x}_{[1]})) + (1 - \pi(\mathbf{x})) \cdot \pi(\mathbf{x}_{[1]}), \\ &= 2\pi(\mathbf{x}) \cdot (1 - \pi(\mathbf{x})) + \epsilon_N(\mathbf{x}), \\ &= 2\eta(\mathbf{x}) \cdot (1 - \eta(\mathbf{x})) + \epsilon_N(\mathbf{x}),\end{aligned}$$

where we defined $\eta(\mathbf{x}) = \min\{\pi(\mathbf{x}), 1 - \pi(\mathbf{x})\}$ (the minimum probability of error on $\mathbf{x}$), and $\epsilon_N(\mathbf{x}) = (2\pi(\mathbf{x}) - 1) \cdot (\pi(\mathbf{x}) - \pi(\mathbf{x}_{[1]}))$. Observe that

$$|\epsilon_N(\mathbf{x})| \leq |\pi(\mathbf{x}) - \pi(\mathbf{x}_{[1]})|,$$

because $0 \leq \pi(\mathbf{x}) \leq 1$. To get $E_{\text{out}}(g_N)$, we take the expectation with respect to $\mathbf{x}$ and use $E^*_{\text{out}} = \mathbb{E}[\eta(\mathbf{x})]$ $\mathbb{E}[\eta(\mathbf{x})^2] \geq \mathbb{E}[\eta(\mathbf{x})]^2 = {E^*_{\text{out}}}^2$:

$$\begin{aligned}E_{\text{out}}(g_N) &= \mathbb{E}\left[\mathbb{P}[g_N(\mathbf{x}) \neq y]\right] \\ &= 2\,\mathbb{E}\left[\eta(\mathbf{x})\right] - 2\,\mathbb{E}\left[\eta^2(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}}[\epsilon_N(\mathbf{x})] \\ &\leq 2E^*_{\text{out}}(1 - E^*_{\text{out}}) + \mathbb{E}_{\mathbf{x}}[\epsilon_N(\mathbf{x})].\end{aligned}$$

Under very general conditions, $\mathbb{E}_{\mathbf{x}}[\epsilon_N(\mathbf{x})] \to 0$ in probability, which implies Theorem 6.1. We don't give a formal proof, but sketch the intuition for why $\epsilon_N(\mathbf{x}) \to 0$. When the data set gets very large ($N \to \infty$), *every* point $\mathbf{x}$ has a nearest neighbor that is close by. That is, $\mathbf{x}_{[1]}(\mathbf{x}) \to \mathbf{x}$ for all $\mathbf{x}$. This is the case if, for example, $P(\mathbf{x})$ has bounded support[6] as shown in Problem 6.9 (more general settings are covered in the literature). By the continuity of $\pi(\mathbf{x})$, if $\mathbf{x}_{[1]} \to \mathbf{x}$ then $\pi(\mathbf{x}_{[1]}) \to \pi(\mathbf{x})$, and since $|\epsilon_N(\mathbf{x})| \leq |\pi(\mathbf{x}_{[1]}) - \pi(\mathbf{x})|$, it follows that $\epsilon_N(\mathbf{x}) \to 0$.[7] The proof also highlights when the nearest neighbor works: a test point must have a nearby neighbor, and the noisy target $\pi(\mathbf{x})$ should be slowly varying so that the y-value at the nearby neighbor is indicative of the y-value at $\mathbf{x}$.

When E^*_{out} is small, $E_{\text{out}}(g)$ is at most twice as bad. When E^*_{out} is large, you are doomed anyway (but on the positive side, you will get a better factor than 2 ☺). It is quite startling that as simple a rule as nearest neighbor is 2-optimal.

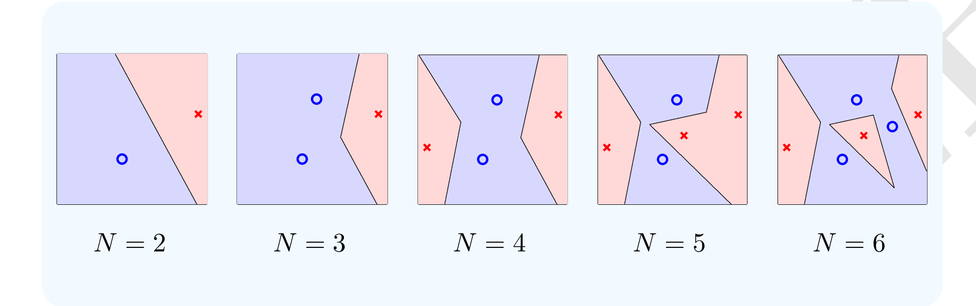
At least half the classification power of the data is in the nearest neighbor.

The rate of convergence to 2-optimal depends on how smooth $\pi(\mathbf{x})$ is, but the convergence itself is never in doubt, relying only on the weak requirement that $\pi(\mathbf{x})$ is continuous. The fact that you get a small E_{out} from a model that can fit any data set has led to the notion that the nearest neighbor rule is somehow 'self-regularizing'. Recall that regularization guides the learning

[6] $\mathbf{x} \in \text{supp}(P)$ if for any $\epsilon > 0$, $\mathbb{P}[B_\epsilon(\mathbf{x})] > 0$ ($B_\epsilon(\mathbf{x})$ is the ball of radius ϵ centered on $\mathbf{x}$).

[7] One technicality remains because we need to interchange the order of limit and expectation: $\lim_{N\to\infty} \mathbb{E}_{\mathbf{x}}[\epsilon(\mathbf{x})] = \mathbb{E}_{\mathbf{x}}[\lim_{N\to\infty} \epsilon(\mathbf{x})] = 0$. For the technically inclined, this can be justified under mild measurability assumptions using dominated convergence.

toward simpler hypotheses, and helps to combat noise, especially when there are fewer data points. This self regularizing in the nearest neighbor algorithm occurs naturally because the final hypothesis is indeed simpler when there are fewer data points. The following examples of the nearest neighbor classifier with increasing number of data points illustrates how the boundary gets more complicated only as you increase N.



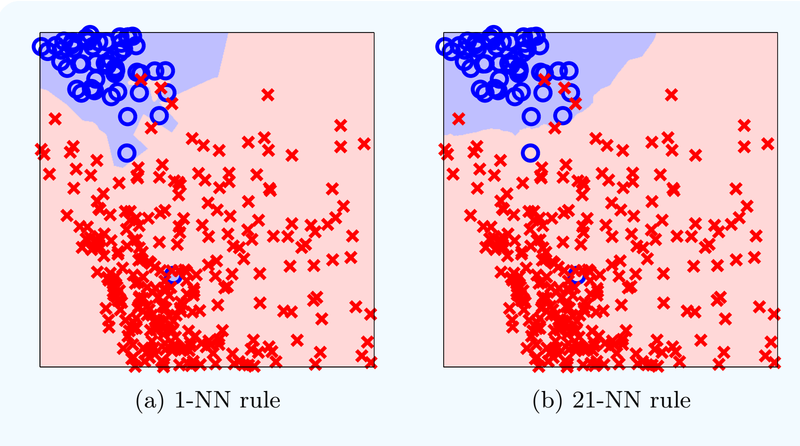
6.2.2 k-Nearest Neighbors (k-NN)

By considering more than just a single neighbor, one obtains the generalization of the nearest neighbor rule to the k-nearest neighbor rule (k-NN). For simplicity, assume that k is odd. The k-NN rule classifies the test point $\mathbf{x}$ according to the majority class among the k nearest data points to $\mathbf{x}$ (the k nearest neighbors). The nearest neighbor rule is the 1-NN rule. For $k > 1$, the final hypothesis is:

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{k} y_{[i]}(\mathbf{x})\right).$$

(k is odd and $y_n = \pm 1$). Figure 6.2 gives a comparison of the nearest neighbor rule with the 21-NN rule on the digits data (blue circles are the digit 1 and all other digits are the red x's). When k is large, the hypothesis is 'simpler'; in particular, when $k = N$ the final hypothesis is a constant equal to the majority class in the data. The parameter k controls the complexity of the resulting hypothesis. Smaller values of k result in a more complex hypothesis, with lower in-sample-error. The nearest neighbor rule always achieves zero in-sample error. When $k > 1$ (see Figure 6.2(b)), the in-sample error is not necessarily zero and we observe that the complexity of the resulting classification boundary drops considerably with higher k.

Three Neighbors is Enough. As with the simple nearest neighbor rule, one can analyze the performance of the k-NN rule (for large N). The next exercise guides the reader through this analysis.



(a) 1-NN rule (b) 21-NN rule

Figure 6.2: The 1-NN and 21-NN rules for classifying a random sample of 500 digits (1 (blue circle) vs all other digits). Note, $21 \approx \sqrt{500}$. For the 1-NN rule, the in-sample error is zero, resulting in a complicated decision boundary with islands of red and blue regions. For the 21-NN rule, the in-sample error is not zero and the decision boundary is 'simpler'.

Exercise 6.3

Fix an odd $k \geq 1$. For $N = 1, 2, \ldots$ and data sets $\{\mathcal{D}_N\}$ of size N, let g_N be the k-NN rule derived from $\mathcal{D}_N$, with out-of-sample error $E_{\text{out}}(g_N)$.

(a) Argue that $E_{\text{out}}(g_N) = \mathbb{E}_{\mathbf{x}}[Q_k(\eta(\mathbf{x}))] + \mathbb{E}_{\mathbf{x}}[\epsilon_N(\mathbf{x})]$ for some error term $\epsilon_N(\mathbf{x})$ which converges to zero, and where

$$Q_k(\eta) = \sum_{i=0}^{(k-1)/2} \binom{k}{i} \left(\eta^{i+1}(1-\eta)^{k-i} + (1-\eta)^{i+1}\eta^{k-i}\right),$$

and $\eta(\mathbf{x}) = \min\{\pi(\mathbf{x}), 1 - \pi(\mathbf{x})\}$.

(b) Plot $Q_k(\eta)$ for $\eta \in [0, \frac{1}{2}]$ and $k = 1, 3, 5$.

(c) Show that for large enough N, with probability at least $1 - \delta$,

$$k = 3: \quad E_{\text{out}}(g_N) \leq E^*_{\text{out}} + 3\,\mathbb{E}\left[\eta^2(\mathbf{x});\right]$$
$$k = 5: \quad E_{\text{out}}(g_N) \leq E^*_{\text{out}} + 10\,\mathbb{E}\left[\eta^3(\mathbf{x})\right].$$

(d) *[Hard]* Show that $E_{\text{out}}(g_N)$ is asymptotically $E^*_{\text{out}}(1 + O(k^{-1/2}))$. *[Hint: Use your plot of Q_k to argue that there is some $a(k)$ such that $Q_k \leq \eta(1 + a(k))$, and show that the best such $a(k)$ is $O(1/\sqrt{k})$.]*

For $k = 3$, as N grows, one finds that the out-of-sample error is nearly optimal:

$$E_{\text{out}}(g_N) \leq E^*_{\text{out}} + 3\,\mathbb{E}\left[\eta^2(\mathbf{x})\right].$$

To interpret this result, suppose that $\eta(\mathbf{x})$ is approximately constant. Then, for $k = 3$, $E_{\text{out}}(g_N) \le E^*_{\text{out}} + 3(E^*_{\text{out}})^2$ (when N is large). The intuition here is that the nearest neighbor classifier makes a mistake if there is an error on the test $\mathbf{x}$ *or* if there are at least 2 mistakes on the three nearest neighbors. The probability of two errors is $O(\eta^2)$ which is dominated by the probability of error on the test point. The 3-NN rule is approximately optimal, converging to E^*_{out} as $E^*_{\text{out}} \to 0$.

To illustrate further, suppose the optimal classifier gives $E^*_{\text{out}} = 1\%$. Then, for large enough N, 3-NN delivers E_{out} of at most 1.03%, which is essentially optimal. The exercise also shows that going to 5-NN improves the asymptotic performance to 1.001%, which is certainly closer to optimal than 3-NN, but hardly worth the effort. Thus, a useful practical guideline is: $k = 3$ nearest neighbors is often enough.

Approximation Versus Generalization. The parameter k controls the approximation-generalization trade off. Choosing k too small leads to too complex a decision rule, which overfits the data; on the other hand, k too large leads to underfitting. Though $k = 3$ works well when E^*_{out} is small, for general situations we need to choose between the different k: each value of k is a different model. We need to pick the best value of k (the best model). That is, we need model selection.

Exercise 6.4

Consider the task of selecting a nearest neighbor rule. What's wrong with the following logic applied to selecting k? (Limits are as $N \to \infty$.)

> Consider the hypothesis set $\mathcal{H}_{\text{NN}}$ with N hypotheses, the k-NN rules using $k = 1, \ldots, N$. Use the in-sample error to choose a value of k which minimizes E_{in}. Using the generalization error bound in Equation (2.1), conclude that $E_{\text{in}} \to E_{\text{out}}$ because $\log N/N \to 0$. Hence conclude that asymptotically, we will be picking the best value of k, based on E_{in} alone.

[Hints: What value of k will be picked? What will E_{in} be? Does your 'hypothesis set' depend on the data?]

Fortunately, there is powerful theory which gives a guideline for selecting k as a function of N; let $k(N)$ be the choice of k, which is chosen to grow as N grows. Note that $1 \le k(N) \le N$.

Theorem 6.2. For $N \to \infty$, if $k(N) \to \infty$ and $k(N)/N \to 0$ then,

$$E_{\text{in}}(g) \to E_{\text{out}}(g) \qquad \text{and} \qquad E_{\text{out}}(g) \to E^*_{\text{out}}.$$

Theorem 6.2 holds under smoothness assumptions on $\pi(\mathbf{x})$ and regularity conditions on $P(\mathbf{x})$ which are very general. The theorem sets $k(N)$ as a function of N and states that as $N \to \infty$, we can recover the optimal classifier. One good choice for $k(N)$ is $\lfloor \sqrt{N} \rfloor$.

We sketch the intuition for Theorem 6.2. For a test point $\mathbf{x}$, consider the distance r to its k-th nearest neighbor. The fraction of points which fall within distance r of $\mathbf{x}$ is k/N which, by assumption, is approaching zero. The only way that the fraction of points within r of $\mathbf{x}$ goes to zero is if r itself tends to zero. That is, *all* the k nearest neighbors to $\mathbf{x}$ are approaching $\mathbf{x}$. Thus, the fraction of these k neighbors with $y = +1$ is approximating $\pi(\mathbf{x})$ and as $k \to \infty$ this fraction will approach $\pi(\mathbf{x})$ by the law of large numbers. The majority vote among these k neighbors will therefore be implementing the optimal classifier f from Exercise 6.2, and hence we will obtain the optimal error E^*_{out}. We can also directly identify the roles played by the two assumptions in the statement of the theorem. The $k/N \to 0$ condition is to ensure that all the k nearest neighbors are close to $\mathbf{x}$; the $k \to \infty$ is to ensure that there are enough close neighbors so that the majority vote is almost always the optimal decision.

Few practitioners would be willing to blindly set $k = \lfloor\sqrt{N}\rfloor$. They would much rather the data spoke for itself. Validation, or cross validation (Chapter 4) are good ways to select k. Validation to select a nearest neighbor classifier would run as follows. Randomly divide the data $\mathcal{D}$ into a training set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$ of sizes $N - K$ and K respectively (the size of the validation set K is not to be confused with k, the number of nearest neighbors to use). Let $\mathcal{H}_{\text{train}} = \{g_1^-, \ldots, g_{N-K}^-\}$ be $N-K$ hypotheses[8] defined by the k-NN rules obtained from $\mathcal{D}_{\text{train}}$, for $k = 1, \ldots, N-K$. Now use $\mathcal{D}_{\text{val}}$ to select one hypothesis g^- from $\mathcal{H}_{\text{train}}$ (the one with minimum validation error). Similarly, one could use cross validation instead of validation.

Exercise 6.5

Consider using validation to select a nearest neighbor rule (hypothesis g^- from $\mathcal{H}_{\text{train}}$). Let g_*^- be the hypothesis in $\mathcal{H}_{\text{train}}$ with minimum E_{out}.

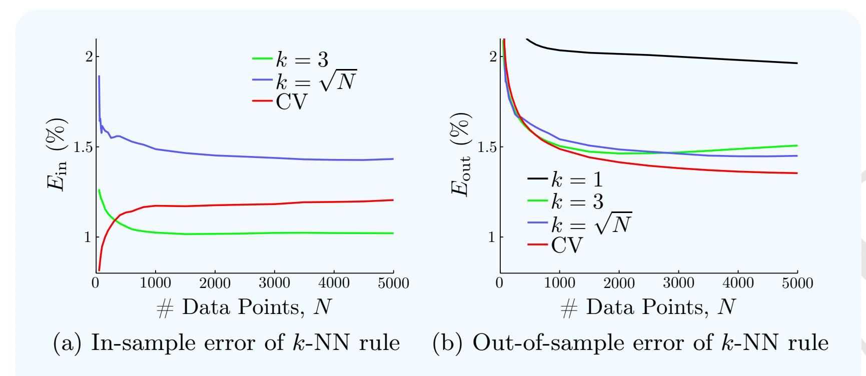
(a) Show that if $K/\log(N-K) \to \infty$ then validation chooses a good hypothesis, $E_{\text{out}}(g^-) \approx E_{\text{out}}(g_*^-)$. Formally state such a result and show it to be true. *[Hint: The statement has to be probabilistic; use the Hoeffding bound and the fact that choosing g^- amounts to selecting a hypothesis from among $N-K$ using a data set of size K.]*

(b) If also $N - K \to \infty$, then show that $E_{\text{out}}(g^-) \to E^*_{\text{out}}$ (validation results in near optimal performance). *[Hint: Use (a) together with Theorem 6.2 which shows that some value of k is good.]*

Note that the selected g^- is not a nearest neighbor rule on the full data $\mathcal{D}$; it is a nearest neighbor rule using data $\mathcal{D}_{\text{train}}$, and k^- neighbors. Would the performance improve if we used the k^--NN rule on the full data set $\mathcal{D}$?

The previous exercise shows that validation works if the validation set size is appropriately chosen, for example, proportional to N. Figure 6.3 illustrates

[8]The g^- notation was introduced in Chapter 4 to denote a hypothesis built from the data set that is missing some data points.



(a) In-sample error of k-NN rule (b) Out-of-sample error of k-NN rule

Figure 6.3: In and out-of-sample errors for various choices of k for the 1 vs all other digits classification task. (a) shows the in-sample error; (b) the out-of-sample error. N points are randomly selected for training, and the remaining for computing the test error. The performance is averaged over 10,000 such training-testing splits. Observe that the 3-NN is almost as good as anything; the in and out-of sample errors for $k = N^{1/2}$ are a close match, as expected from Theorem 6.2. The cross validation in-sample error is approaching the out-of-sample error. For $k = 1$ the in-sample error is zero, and is not shown.

our main conclusions on the digits data using $k = \lfloor\sqrt{N}\rfloor$, k chosen using 10-fold cross validation and k fixed at $1, 3$.

6.2.3 Improving the Efficiency of Nearest Neighbor

The simplicity of the nearest neighbor rule is both a virtue and a vice. There's no training cost, but we pay for this when predicting on a test input during deployment. For a test point $\mathbf{x}$, we need to compute the k nearest neighbors. This means we need access to all the data, a memory requirement of Nd and a computational complexity of $O(Nd + N \log k)$ (using appropriate data structures to compute N distances and pick the smallest k). This can be excessive as the next exercise illustrates.

Exercise 6.6

We want to select the value of $k = 1, 3, 5, \ldots, 2\lfloor \frac{N+1}{2} \rfloor - 1$ using 10-fold cross validation. Show that the running time is $O(N^3 d + N^3 \log N)$

The basic approach to addressing these issues is to preprocess the data in some way. This can help by either storing the data in a specialized data structure so that the nearest neighbor queries are more efficient, or by reducing the amount of data that needs to be kept (which helps with the memory and computational

requirements when deploying). These are active research areas, and we discuss the basic approaches here.

Data Condensing The goal of data condensing is to reduce the data set size while making sure the retained data set somehow matches the original full data set. We denote the condensed data set by S. We will only consider the case when S is a subset of $\mathcal{D}$.[9]

Fix k, and for the data set S, let g_S be the k-NN rule derived from S and, similarly, g_D is the k-NN rule derived from the full data set $\mathcal{D}$. The condensed data set S is *consistent* with $\mathcal{D}$ if

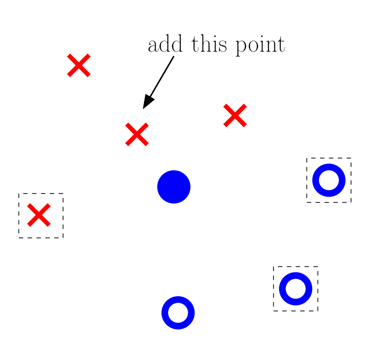
$$g_S(\mathbf{x}) = g_{\mathcal{D}}(\mathbf{x}) \qquad \forall \mathbf{x} \in \mathbb{R}^d.$$

This is a strong requirement, and there are computationally expensive algorithms which can achieve this (see Problem 6.11). A weaker requirement is that the condensed data only be *training set consistent*, which means that the condensed data and the full training data give the same classifications on the training data points,

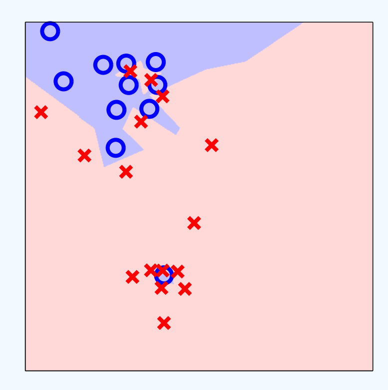
$$g_S(\mathbf{x}_n) = g_{\mathcal{D}}(\mathbf{x}_n) \qquad \forall \mathbf{x}_n \in \mathcal{D}.$$

This means that, as far as the training points are concerned, the rule derived from S is the same as the rule derived from $\mathcal{D}$. Note that training set consistent does not mean $E_{\text{in}}(g_S) = 0$. For starters, when $k > 1$, even the full data $\mathcal{D}$ may not achieve $E_{\text{in}}(g_{\mathcal{D}}) = 0$.

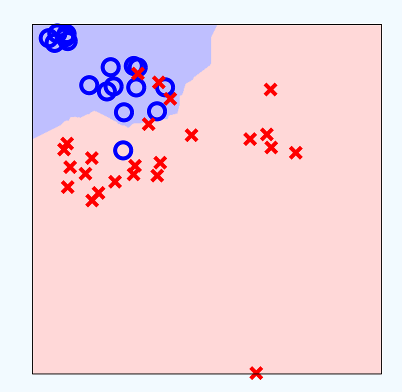
The condensed nearest neighbor algorithm (CNN) is a very simple heuristic that results in a training set consistent subset of the data. We illustrate with a data set of 8 points on the right, setting k to 3. Initialize the working set S randomly with k data points from $\mathcal{D}$ (the boxed points illustrated on the right). As long as S is not training set consistent, augment S with a data point not in S as follows. Let $\mathbf{x}_*$ be *any* data point for which $g_S(\mathbf{x}_*) \neq g_{\mathcal{D}}(\mathbf{x}_*)$, such as the solid blue circle in the example. This solid blue point is classified blue by S (in fact, every point is classified blue by S because S only contains three points, two of which are blue); but, it is classified red by $\mathcal{D}$ (because two of its three nearest points in $\mathcal{D}$ are red). So $g_S(\mathbf{x}_*) \neq g_{\mathcal{D}}(\mathbf{x}_*)$. Let $\mathbf{x}'$ be the nearest data point to $\mathbf{x}_*$ which is not in S *and* has class $g_{\mathcal{D}}(\mathbf{x}_*)$. Note that $\mathbf{x}_*$ could be in S. The reader can easily verify that such an $\mathbf{x}'$ must exist because $\mathbf{x}_*$ is inconsistently classified by S. Add $\mathbf{x}'$ to S. Continue adding to S in this way until S is training set consistent. (Note that $\mathcal{D}$ is training set consistent, by definition.)



[9] More generally, if S contains arbitrary points, S is called a condensed set of *prototypes*.



(a) Condensed data for 1-NN



(b) Condensed data for 21-NN

Figure 6.4: The condensed data and resulting classifiers from running the CNN heuristic to condense the 500 digits data points used in Figure 6.2. (a) Condensation to 27 data points for 1-NN training set consistent (b) Condensation to 40 data points for 21-NN training set consistent.

The CNN algorithm must terminate after at most N steps, and when it terminates, the resulting set S must be training set consistent. The CNN algorithm delivers good condensation in practice as illustrated by the condensed data in Figure 6.4 as compared with the full data in Figure 6.2.

Exercise 6.7

Show the following properties of the CNN heuristic. Let S be the current set of points selected by the heuristic.

(a) If S is not training set consistent, and if $\mathbf{x}_*$ is a point which is not training set consistent, show that the CNN heuristic will always find a point to add to S.

(b) Show that the point added will 'help' with the classification of $\mathbf{x}_*$ by S; it suffices to show that the new k nearest neighbors to $\mathbf{x}_*$ in S will contain the point added.

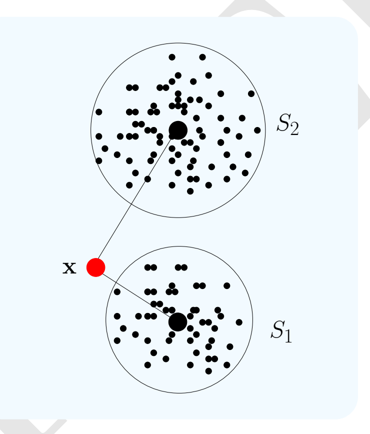
(c) Show that after at most $N - k$ iterations the CNN heuristic must terminate with a training set consistent S.

The CNN algorithm will generally not produce a training consistent set of minimum cardinality. Further, the condensed set can vary depending on the order in which data points are considered, and the process is quite computationally expensive; but, at least it is done only once, and it saves every time a new point is to be classified.

We should distinguish data condensing from *data editing*. The former has a purely computational motive, to reduce the size of the data set while keeping the decision boundary nearly unchanged. The latter edits the data set to

improve prediction performance. This typically means remove the points that you think are noisy (see Chapter 9). One way to determine if a point is noisy is if it disagrees with the majority class in its k-neighborhood. Such data points are removed from the data and the remaining data are used with the 1-NN rule. A very similar effect can be accomplished by just using the k-NN rule with the original data.

Efficient Search for the Nearest Neighbors The other main approach to alleviating the computational burden is to preprocess the data in some way so that nearest neighbors can be found quickly. There is a large and growing volume of literature in this arena. Our goal here is to describe the basic idea with a simple technique.



Suppose we want to search for the nearest neighbor to the point $\mathbf{x}$ (red). Assume that the data is partitioned into clusters (or *branches*). In our illustration, we have two clusters S_1 and S_2. Each cluster has a 'center' $\boldsymbol{\mu}_j$ and a radius r_j. First search the cluster whose center is closest to the query point $\mathbf{x}$. In the example, we search cluster S_1 because $\|\mathbf{x}-\boldsymbol{\mu}_1\| \leq \|\mathbf{x}-\boldsymbol{\mu}_2\|$. Suppose that the nearest neighbor to $\mathbf{x}$ in S_1 is $\hat{\mathbf{x}}_{[1]}$, which is potentially $\mathbf{x}_{[1]}$, the nearest point to $\mathbf{x}$ in all the data. For any $\mathbf{x}_n \in S_2$, by the triangle inequality, $\|\mathbf{x}-\mathbf{x}_n\| \geq \|\mathbf{x}-\boldsymbol{\mu}_2\| - r_2$. Thus, if the *bound* condition

$$\|\mathbf{x}-\hat{\mathbf{x}}_{[1]}\| \leq \|\mathbf{x}-\boldsymbol{\mu}_2\| - r_2$$

holds, then we are done (we don't need to search S_2), and can claim that $\hat{\mathbf{x}}_{[1]}$ is a nearest neighbor ($\mathbf{x}_{[1]} = \hat{\mathbf{x}}_{[1]}$). We gain a computational saving if the bound condition holds. This approach is called a *branch and bound* technique for finding the nearest neighbor. Let's informally consider when we are likely to satisfy the bound condition, and hence obtain computational saving. By the triangle inequality, $\|\mathbf{x}-\hat{\mathbf{x}}_{[1]}\| \leq \|\mathbf{x}-\boldsymbol{\mu}_1\| + r_1$, so, if $\|\mathbf{x}-\boldsymbol{\mu}_1\| + r_1 \leq \|\mathbf{x}-\boldsymbol{\mu}_2\| - r_2$, then the bound condition is certainly satisfied. To interpret this sufficient condition, consider the case when $\mathbf{x}$ is much closer to $\boldsymbol{\mu}_1$, in which case $\|\mathbf{x}-\boldsymbol{\mu}_1\| \approx 0$ and $\|\mathbf{x}-\boldsymbol{\mu}_2\| \approx \|\boldsymbol{\mu}_1-\boldsymbol{\mu}_2\|$. Then we want

$$r_1 + r_2 \ll \|\boldsymbol{\mu}_1-\boldsymbol{\mu}_2\|$$

to ensure the bound holds. In words, the radii of the clusters (the *within cluster spread*) should be much smaller than the distance between clusters (the *between cluster spread*). A good clustering algorithm should produce exactly such clusters, with small within cluster spread and large between cluster spread.

There is nothing to prevent us from applying the branch and bound technique recursively within each cluster. So, for example, to search for the nearest neighbor inside S_1, we can assume that S_1 has been partitioned into (say) two clusters S_{11} and S_{12}, and so on.

Exercise 6.8

(a) Give an algorithmic pseudo code for the recursive branch and bound search for the nearest neighbor, assuming that every cluster with more than 2 points is split into 2 sub-clusters.

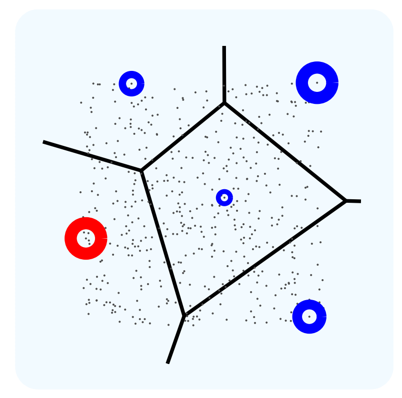
(b) Assume the sub-clusters of a cluster are balanced, i.e. contain exactly half the points in the cluster. What is the maximum depth of the recursive search for a nearest neighbor. (Assume the number of data points is a power of 2).

(c) Assume balanced sub-clusters and that the bound condition always holds. Show that the time to find the nearest neighbor is $O(d \log N)$.

(d) How would you apply the branch and bound technique to finding the k-nearest neighbors as opposed to the single nearest neighbor?

The computational savings of the branch and bound algorithm depend on how likely it is that the bound condition is met, which depends on how good the partitioning is. We observe that the clusters should be balanced (so that the recursion depth is not large), well separated and have small radii (so the bound condition will often hold). A very simple heuristic for getting such a good partition into M clusters is based on a clustering algorithm we will discuss later. Here is the main idea.

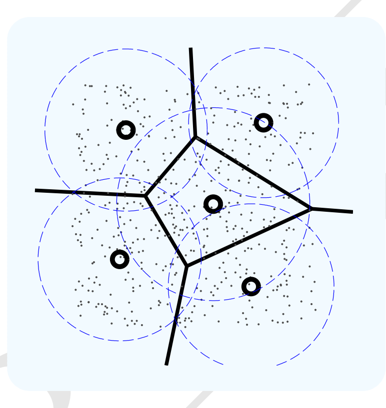
First create a set of M well separated centers for the clusters; we recommend a simple greedy approach. Start by taking an arbitrary data point as a center. Now, as a second center, add the point furthest from the first center; to get each new center, keep adding the point furthest from the current set of centers until you have M centers (the distance of a point from a set is the distance to its nearest neighbor in the set). In the example, the red point is the first (random) center added; the largest blue point is added next and so on. This greedy algorithm can be implemented in $O(MNd)$ time using appropriate data structures. The partition is inferred from the Voronoi regions of each center: all points in a center's Voronoi region belong to that center's cluster. The center of each cluster is redefined as the average data point in the cluster, and its radius is the maximum



distance from the center to a point in the cluster,

$$\boldsymbol{\mu}_j = \frac{1}{|S_j|} \sum_{\mathbf{x}_n \in S_j} \mathbf{x}_n; \qquad r_j = \max_{\mathbf{x}_n \in S_j} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|.$$

We can improve the clusters by obtaining new Voronoi regions defined by these new centers $\boldsymbol{\mu}_j$, and then updating the centers and radii appropriately for these new clusters. Naturally, we can continue this process iteratively (we introduce this algorithm later in the chapter as Lloyd's algorithm for clustering). The main goal here is computational efficiency, and the perfect partition is not necessary. The partition, centers and radii after one refinement are illustrated on the right. Note that though the clusters are disjoint, the spheres enclosing the clusters need not be.



The clusters, together with $\boldsymbol{\mu}_j$ and r_j can be computed in $O(MNd)$ time. If the depth of the partitioning is $O(\log N)$, which is the case if the clusters are nearly balanced, then the total run time to compute all the clusters at every level is $O(MNd \log N)$. In Problem 6.16 you can experiment with the performance gains from such simple branch and bound techniques.

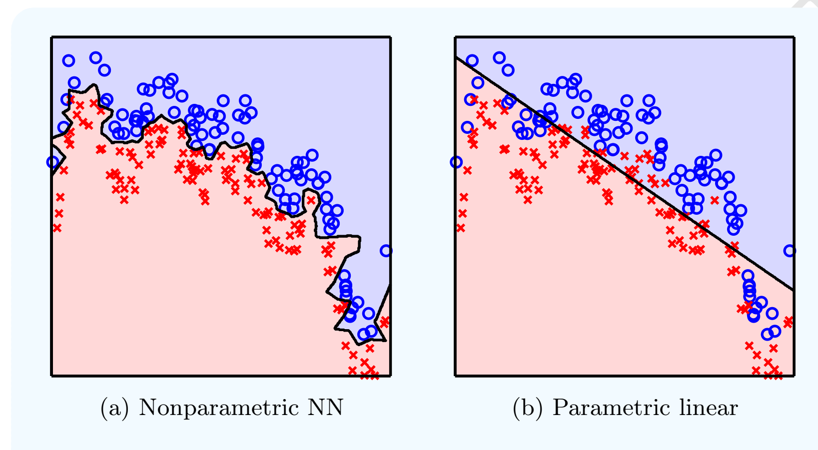
6.2.4 Nearest Neighbor is Nonparametric

There is some debate in the literature on *parametric* versus *nonparametric* learning models. Having covered linear models, and nearest neighbor, we are in a position to articulate some of this discussion. We begin by identifying some of the defining characteristics of parametric versus nonparametric models.

Nonparametric methods have no *explicit* parameters to be learned from data; they are, in some sense, general and expressive or flexible; and, they typically store the training data, which are then used during deployment to predict on a test point. Parametric methods, on the other hand, explicitly learn certain parameters from data. These parameters specify the final hypothesis from a typically 'restrictive' *parameterized* functional form; the learned parameters are sufficient for prediction, i.e. the parameters summarize the data which is then not needed later during deployment.

To further highlight the distinction between these two types of techniques, consider the nearest neighbor rule versus the linear model. The nearest neighbor rule is the prototypical nonparametric method. There are no parameters to be learned. Yet, the nonparametric nearest neighbor method is flexible and general: the final decision rule can be very complex, or not, molding its shape to the data. The linear model, which we saw in Chapter 3 is the prototypical

parametric method. In d dimensions, there are $d+1$ parameters (the weights and bias) which need to be learned from the data. These parameters summarize the data, and after learning, the data can be discarded: prediction only needs the learned parameters. The linear model is quite rigid, in that no matter what the data, you will always get a linear separator as g. Figure 6.5 illustrates this difference in flexibility on a toy example.



(a) Nonparametric NN (b) Parametric linear

Figure 6.5: The decision boundary of the 'flexible' nonparametric nearest neighbor rule molds itself to the data, whereas the 'rigid' parametric linear model will always give a linear separator.

The k-nearest neighbor method would also be considered nonparametric (once the 'parameter' k has been specified). Theorem 6.2 is an example of a general convergence result.[10] Under mild regularity conditions, no matter what the target f is, we can recover it as $N \to \infty$, provided that k is chosen appropriately. That's quite a powerful statement about such a simple learning model applied to learning a general f. Such convergence results under mild assumptions on f are a trademark of nonparametric methods. This has led to the folklore that nonparametric methods are, in some sense, more powerful than their cousins the parametric methods: for the parametric linear model, *only if* the target f happens to be in the linearly parameterized hypothesis set, can one get such convergence to f with larger N.

To complicate the distinction between the two methods, let's look at the non-linear feature transform (e.g. the polynomial transform). As the polynomial order increases, the number of parameters to be estimated increases and $\mathcal{H}$, the hypothesis set, gets more and more expressive. It is possible to choose the polynomial order to increase with N, but not too quickly so that $\mathcal{H}$ gets more and more expressive, eventually capturing any fixed target and yet

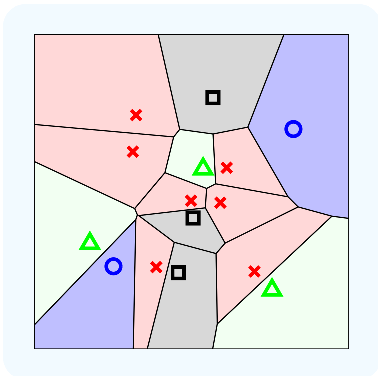
[10] For the technically oriented, it establishes the *universal consistency* of the k-NN rule.

the number of parameters grows slowly enough that it is possible to effectively learn them from the data. Such a model has properties of both the non-parametric and parametric settings: you still need to learn parameters (like parametric models) but the hypothesis set is getting more and more expressive so that you do get convergence to f. As a result, such techniques have sometimes been called parametric, and sometimes nonparametric, neither being a satisfactory adjective; we prefer the term *semi-parametric*. Many techniques are semi-parametric, for example Neural Networks (see Chapter 7).

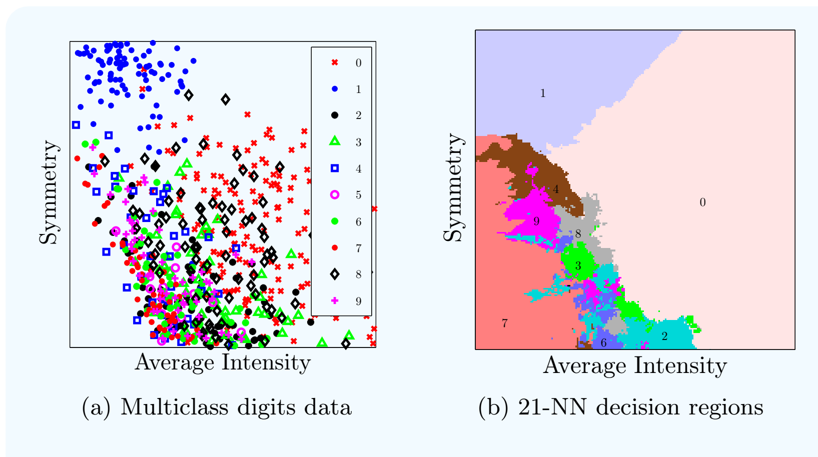
In applications, the purely parametric model (like the linear model) has become synonymous with 'specialized' in that you have thought very carefully about the features and the form of the model and you are confident that the specialized linear model will work. To take another example from interest rate prediction, a well known specialized parametric model is the multi-factor Vasicek model. If we had no idea what the appropriate model is for interest rate prediction, then rather than commit to some specialized model, one might use a 'generic' nonparametric or semi-parametric model such as k-NN or neural networks. In this setting, the use of the more generic model, by being more flexible, accommodates our inability to pin down the specific form of the target function. We stress that the availability of 'generic' nonparametric and semi-parametric models is not a license to stop thinking about the right features and form of the model; it is just a backstop that is available when the target is not easy to explicitly model parametrically.

6.2.5 Multiclass Data

The nearest neighbor method is an elegant platform to study an issue which, unto now, we have ignored. In our digits data, there are not 2 classes (± 1), but 10 classes ($0, 1, \dots, 9$). This is a *multiclass* problem. Several approaches to the multiclass problem decompose it into a sequence of 2 class problems. For example, first classify between '1' and 'not 1'; if you classify as '1' then you are done. Otherwise determine '2' versus 'not 2'; this process continues until you have narrowed down the digit. Several complications arise in determining what is the best sequence of 2 class problems; and, they need not be one versus all (they can be any subset versus its complement).



The nearest neighbor method offers a simple solution. The class of the test point $\mathbf{x}$ is simply the class of its nearest neighbor $\mathbf{x}_{[1]}$. One can also generalize to the k-nearest neighbor rule: the class of $\mathbf{x}$ is the majority class among the k-nearest neighbors $\mathbf{x}_{[1]}, \dots, \mathbf{x}_{[k]}$, breaking ties randomly.



(a) Multiclass digits data (b) 21-NN decision regions

Figure 6.6: 21-NN decision classifier for the multiclass digits data.

Exercise 6.9

With C classes labeled $1, \ldots, C$, define $\pi_c(\mathbf{x}) = \mathbb{P}[c|\mathbf{x}]$ (the probability to observe class c given $\mathbf{x}$, analogous to $\pi(\mathbf{x})$). Let $\eta(\mathbf{x}) = 1 - \max_c \pi_c(\mathbf{x})$.

(a) Define a target $f(\mathbf{x}) = \text{argmax}_c\, \pi_c(\mathbf{x})$. Show that, on a test point $\mathbf{x}$, f attains the minimum possible error probability of

$$e(f(\mathbf{x})) = \mathbb{P}[f(\mathbf{x}) \neq y] = \eta(\mathbf{x}).$$

(b) Show that for the nearest neighbor rule ($k = 1$), with high probability, the final hypothesis g_N achieves an error on the test point $\mathbf{x}$ that is

$$e(g_N(\mathbf{x})) \stackrel{N\to\infty}{\longrightarrow} \sum_{c=1}^{C} \pi_c(\mathbf{x})(1 - \pi_c(\mathbf{x})).$$

(c) Hence, show that for large enough N, with high probability,

$$E_{\text{out}}(g_N) \leq 2E^*_{\text{out}} - \frac{C}{C-1}(E^*_{\text{out}})^2.$$

[Hint: Show that $\sum_i a_i^2 \geq a_1^2 + \frac{(1-a_1)^2}{C-1}$ for any $a_1 \geq \cdots \geq a_C \geq 0$ and $\sum_i a_i = 1$,]

The previous exercise shows that even for the multiclass problem, the simple nearest neighbor is at most a factor of 2 from optimal. One can also obtain a universal consistency result as in Theorem 6.2. The result of running a 21-nearest neighbor rule on the digits data is illustrated in Figure 6.6.

The Confusion Matrix. The probability of misclassification which we discussed in the 2-class problem can be generalized to a *confusion matrix* which

True	Predicted										
	0	1	2	3	4	5	6	7	8	9	
0	**13.5**	0.5	0.5	1	0	0.5	0	0	0.5	0	16.5
1	0.5	**13.5**	0	0	0	0	0	0	0	0	14
2	0.5	0	**3.5**	1	1	1.5	1	1	0	0.5	10
3	2.5	0	1.5	**2**	0.5	0.5	0.5	0.5	0.5	1	9.5
4	0.5	0	1	0.5	**1.5**	0.5	1	2	0	1.5	8.5
5	0.5	0	2.5	1	0.5	**1.5**	1	1	0	0.5	7.5
6	0.5	0	2	1	1	1	**1**	1	0	1	8.5
7	0	0	1.5	0.5	1.5	0.5	1	**3**	0	1	9
8	3.5	0	0.5	1	0.5	0.5	0.5	0	**0.5**	1	8
9	0.5	0	1	1	1	0.5	1	1	0.5	**2**	8.5
	22.5	14	14	9	7.5	7	7	9.5	2	8.5	100

Table 6.1: Out-of-sample confusion matrix for the 21-NN rule on the 10-class digits problem (all numbers are in %). The bold entries along the diagonal are the correct classifications. Practitioners sometimes work with the class conditional confusion matrix which is $\mathbb{P}[c_2|c_1]$, the probability that g classifies c_2 *given* that the true class is c_1. The class conditional confusion matrix is obtained from the confusion matrix by dividing each row by the sum of the elements in the row. Sometimes the class conditional confusion matrix is easier to interpret because the ideal case is a diagonal matrix with all diagonals equal to 100%.

is a better representation of the performance, especially when there are multiple classes. The confusion matrix tells us not only the probability of error, but also the nature of the error, namely which classes are confused with each other. Analogous to the misclassification error, one has the in-sample and out-of-sample confusion matrices. The in-sample (resp. out-of-sample) confusion matrix is a $C \times C$ matrix which measures how often one class is classified as another. So, the out-of-sample confusion matrix is

$$\begin{aligned} \mathrm{E}_{\text{out}}(c_1, c_2) &= \mathbb{P}[\text{class } c_1 \text{ is classified as } c_2 \text{ by } g] \\ &= \mathbb{E}_{\mathbf{x}}\left[\pi_{c_1}(\mathbf{x})\, [\![g(\mathbf{x}) = c_2]\!]\right] \\ &= \int d\mathbf{x}\ \ P(\mathbf{x})\pi_{c_1}(\mathbf{x})\, [\![g(\mathbf{x}) = c_2]\!]\,. \end{aligned}$$

where $[\![\cdot]\!]$ is the indicator function which equals 1 when its argument is true. Similarly, we can define the in-sample confusion matrix,

$$E_{\text{in}}(c_1, c_2) = \frac{1}{N} \sum_{n: y_n = c_1} [\![g(\mathbf{x}_n) = c_2]\!]\,,$$

where the sum is over all data points with classification c_1, and the summand counts the number of those cases that g classifies as c_2. The out-of-sample confusion matrix obtained by running the 21-NN rule on the 10-class digits

data is shown in Table 6.1. The sum of the diagonal elements gives the probability of correct classification, which is about 42%. From Table 6.1, we can easily identify some of the digits which are commonly confused, for example 8 is often classified as 0. In comparison, for the two class problem of classifying digit 1 versus all others, the success was upwards of 98%. For the multiclass problem, random performance would achieve a success rate of 10%, so the performance is significantly above random; however, it is significantly below the 2-class version; multiclass problems are typically much harder than the two class problem. Though symmetry and intensity are two features that have enough information to distinguish between 1 versus the other digits, they are not powerful enough to solve the multiclass problem. Better features would certainly help. One can also gain by breaking up the problem into a sequence of 2-class tasks and tailoring the features to each 2-class problem using domain knowledge.

6.2.6 Nearest Neighbor for Regression.

With multiclass data, the natural way to combine the classes of the nearest neighbors to obtain the class of a test input is by using some form of majority voting procedure. When the output is a real number ($y \in \mathbb{R}$), the natural way to combine the outputs of the nearest neighbors is using some form of averaging. The simplest way to extend the k-nearest neighbor algorithm to regression is to take the average of the target values from the k-nearest neighbors:

$$g(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} y_{[i]}(\mathbf{x}).$$

There are no explicit parameters being learned, and so this is a nonparametric regression technique. Figure 6.7 illustrates the k-NN technique for regression using a toy data set generated by the target function in light gray.

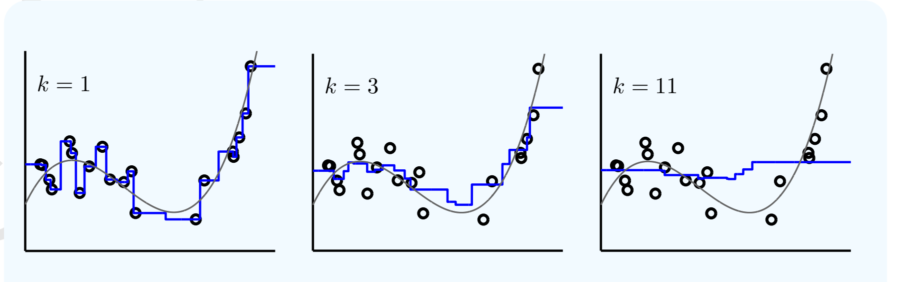


Figure 6.7: k-NN for regression on a noisy toy data set with $N = 20$.

Exercise 6.10

You are using k-NN for regression (Figure 6.7).

(a) Show that E_{in} is zero when $k = 1$.

(b) Why is the final hypothesis not smooth, making step-like jumps?

(c) How does the choice of k control how well g approximates f? (Consider the cases k too small and k too large.)

(d) How does the final hypothesis g behave when $x \to \pm\infty$.

Outputting a Probability with k-NN. Recall that for binary classification, we took $\text{sign}(g(\mathbf{x}))$ as the final hypothesis. We can also get a natural extension of the nearest neighbor technique outputting a probability. The final hypothesis is

$$g(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} [\![y_{[i]} = +1]\!],$$

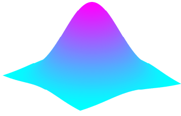
which is just the fraction of the k nearest neighbors that are classified $+1$.

For both regression and logistic regression, as with classification, if $N \to \infty$ and $k \to \infty$ with $k/N \to 0$, then, under mild assumptions, you will recover a close approximation to the target function: $g(\mathbf{x}) \to f(\mathbf{x})$ (as N grows). This convergence is true even if the data (y values) are noisy, with bounded noise variance, because the averaging can overcome that noise when k gets large.

6.3 Radial Basis Functions

In the k-nearest neighbor rule, only k neighbors influence the prediction at a test point $\mathbf{x}$. Rather than fix the value of k, the radial basis function (RBF) technique allows all data points to contribute to $g(\mathbf{x})$, but not equally. Data points further from $\mathbf{x}$ contribute less; a radial basis function (or kernel) ϕ quantifies how the contribution decays as the distance increases. The contribution of $\mathbf{x}_n$ to the classification at $\mathbf{x}$ will be proportional to $\phi(\|\mathbf{x} - \mathbf{x}_n\|)$, and so the properties of a typical kernel are that it is positive and non-increasing. The most popular kernel is the Gaussian kernel,

$$\phi(z) = e^{-\frac{1}{2}z^2}.$$



Another common kernel is the window kernel,

$$\phi(z) = \begin{cases} 1 & z \le 1, \\ 0 & z > 1, \end{cases}$$

It is common to normalize the kernel so that it integrates to 1 over $\mathbb{R}^d$. For the Gaussian kernel, the normalization constant is $(2\pi)^{-d/2}$; for the window kernel, the normalization constant is $\pi^{-d/2}\Gamma(\frac{1}{2}d+1)$, where $\Gamma(\cdot)$ is the Gamma function. The normalization constant is not important for RBFs, but for density estimation (see later), the normalization will be needed.

One final ingredient is the scale r which specifies the width of the kernel. The scale determines the 'unit' of length against which distances are measured. If $\|\mathbf{x} - \mathbf{x}_n\|$ is small relative to r, then $\mathbf{x}_n$ will have a significant influence on the value of $g(\mathbf{x})$. We denote the influence of $\mathbf{x}_n$ at $\mathbf{x}$ by $\alpha_n(\mathbf{x})$,

$$\alpha_n(\mathbf{x}) = \phi\left(\frac{\|\mathbf{x} - \mathbf{x}_n\|}{r}\right).$$

The 'radial' in RBF is because the influence only depends on the radial distance $\|\mathbf{x} - \mathbf{x}_n\|$. We compute $g(\mathbf{x})$ as the weighted sum of the y-values:

$$g(\mathbf{x}) = \frac{\sum_{n=1}^N \alpha_n(\mathbf{x}) y_n}{\sum_{m=1}^N \alpha_m(\mathbf{x})}, \tag{6.1}$$

where the sum in the denominator ensures that the sum of the weights is 1. The hypothesis (6.1) is the direct analog of the nearest neighbor rule for regression. For classification, the final output is $\text{sign}(g(\mathbf{x}))$; and, for logistic regression the output is $\theta(g(\mathbf{x}))$, where $\theta(s) = e^s/(1+e^s)$.

The scale parameter r determines the relative emphasis on nearby points. The smaller the scale parameter, the more emphasis is placed on the nearest points. Figure 6.8 illustrates how the final hypothesis depends on r and the next exercise shows that when $r \to 0$, the RBF is the nearest neighbor rule.

Exercise 6.11

When $r \to 0$, show that for the Gaussian kernel, the RBF final hypothesis is $g(\mathbf{x}) = y_{[1]}$, the same as the nearest neighbor rule.

[Hint: $g(\mathbf{x}) = \dfrac{\sum_{n=1}^N y_{[n]}\alpha_{[n]}/\alpha_{[1]}}{\sum_{m=1}^N \alpha_{[m]}/\alpha_{[1]}}$ *and show that* $\alpha_{[n]}/\alpha_{[1]} \to 0$ *for* $n \neq 1$.*]*

As r gets large, the kernel width gets larger and more of the data points contribute to the value of $g(\mathbf{x})$. As a result, the final hypothesis gets smoother. The scale parameter r plays a similar role to the number of neighbors k in the k-NN rule; in fact, the RBF technique with the window kernel is sometimes

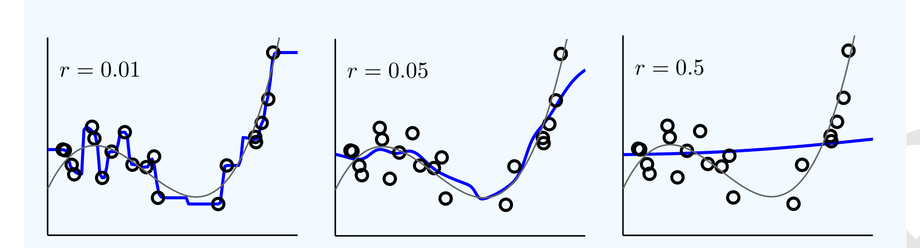


Figure 6.8: The nonparametric RBF using the same noisy data from Figure 6.7. Too small an r results in a complex hypothesis that overfits. Too large an r results in an excessively smooth hypothesis that underfits.

called the r-nearest neighbor rule, because it uniformly weights all neighbors within distance r of $\mathbf{x}$. One way to choose r is using cross validation. Another is to use Theorem 6.2 as a guide. Roughly speaking, the neighbors within distance r of $\mathbf{x}$ contribute to $g(\mathbf{x})$. The volume of this region is order of r^d, and so the number of points in this region is order of Nr^d; thus, using scale parameter r effectively corresponds to using a number of neighbors k that is order of Nr^d. Since we want $k/N \to \infty$ and $k \to \infty$ as N grows, one effective choice for r is $(\sqrt[2d]{N})^{-1}$, which corresponds to $k \approx \sqrt{N}$.

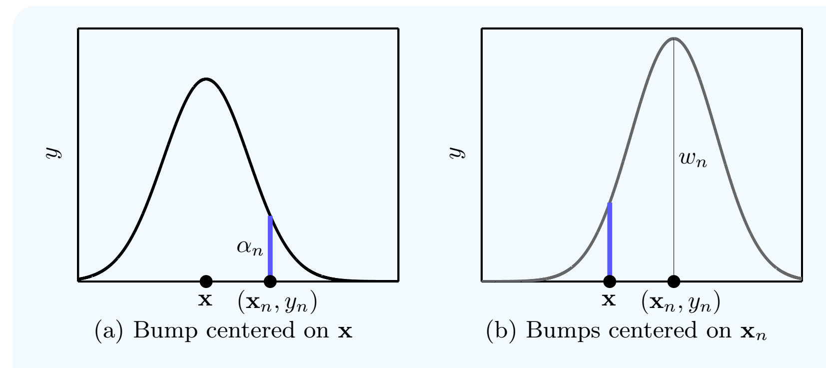
6.3.1 Radial Basis Function Networks

There are two ways to interpret the RBF in equation (6.1). The first is as a weighted sum of y-values as presented in equation (6.1), where the weights are α_n. This corresponds to centering a single kernel, or a bump, at $\mathbf{x}$. This bump decays as you move away from $\mathbf{x}$ and the value the bump attains at $\mathbf{x}_n$ determines the weight α_n (see Figure 6.9(a)). The alternative view is to rewrite Equation (6.1) as

$$g(\mathbf{x}) = \sum_{n=1}^{N} w_n(\mathbf{x})\phi\left(\frac{\|\mathbf{x}-\mathbf{x}_n\|}{r}\right), \tag{6.2}$$

where $w_n(\mathbf{x}) = y_n / \sum_{i=1}^{N} \phi\left(\|\mathbf{x}-\mathbf{x}_i\|/r\right)$. This version corresponds to centering a bump at every $\mathbf{x}_n$. The bump at $\mathbf{x}_n$ has a height $w_n(\mathbf{x})$. As you move away from $\mathbf{x}_n$, the bump decays, and the value the bump attains at $\mathbf{x}$ is the contribution of $\mathbf{x}_n$ to $g(\mathbf{x})$ (see Figure 6.9(b)).

With the second interpretation, $g(\mathbf{x})$ is the sum of N bumps of different heights, where each bump is centered on a data point. The width of each bump is determined by r. The height of the bump centered on $\mathbf{x}_n$ is $w_n(\mathbf{x})$, which varies depending on the test point. If we fix the bump heights to w_n, independent of the test point, then the same bumps centered on the data points apply to every test point $\mathbf{x}$ and the functional form simplifies. Define



(a) Bump centered on $\mathbf{x}$ (b) Bumps centered on $\mathbf{x}_n$

Figure 6.9: Different views of the RBF. (a) $g(\mathbf{x})$ is a weighted sum of y_n with weights α_n determined by a bump centered on $\mathbf{x}$. (b) $g(\mathbf{x})$ is a sum of bumps, one on each $\mathbf{x}_n$ having height w_n.

$\Phi_n(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_n\|/r)$. Then our hypotheses have the form

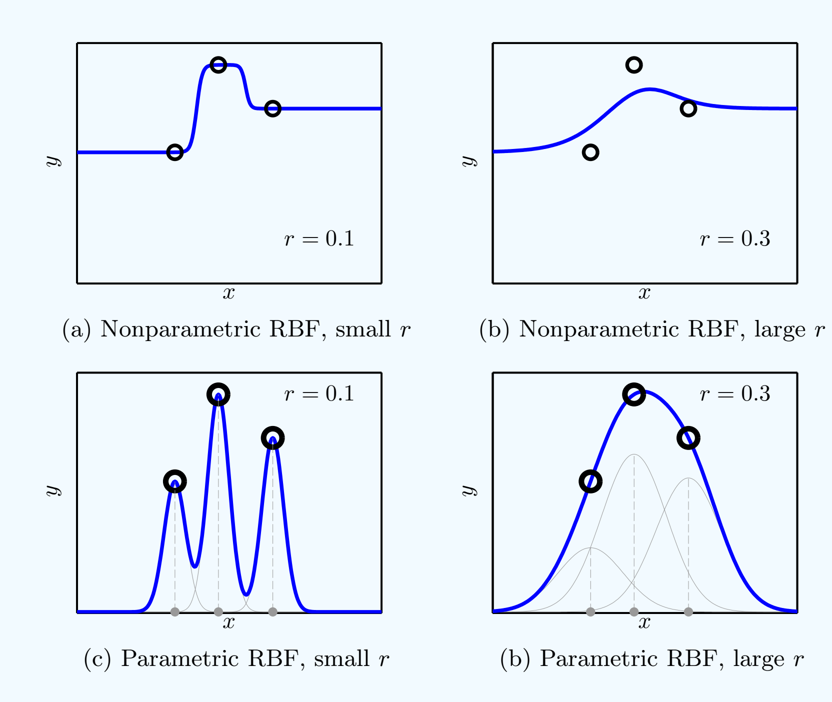
$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \Phi_n(\mathbf{x}), \tag{6.3}$$

where now w_n are constants (parameters) that we get to choose. We switched over to $h(\mathbf{x})$ because as of yet, we do not know what the final hypothesis g is until we specify the weights $w_1, \ldots, w_N$, whereas in (6.2), the weights $w_n(\mathbf{x})$ are specified. Observe that (6.3) is just a linear model $h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{z}$ with the nonlinear transform to an N-dimensional space given by

$$\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \Phi_1(\mathbf{x}) \\ \vdots \\ \Phi_N(\mathbf{x}) \end{bmatrix}.$$

The nonlinear transform is determined by the kernel ϕ and the data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Since the nonlinear transform is obtained by placing a kernel bump on each data point, these nonlinear transforms are often called *local* basis functions. Notice that the nonlinear transform is not known ahead of time. It is only known once the data points are seen.

In the nonparametric RBF, the weights $w_n(\mathbf{x})$ were specified by the data and there was nothing to learn. Now that w_n are parameters, we need to learn their values (the new technique is parametric). As we did with linear models, the natural way to set the parameters w_n is to fit the data by minimizing the in-sample error. Since we have N parameters, we should be able to fit the data exactly. Figure 6.10 illustrates the parametric RBF as compared to the nonparametric RBF, highlighting some of the differences between taking $g(\mathbf{x})$ as the weighted sum of y-values scaled by a bump at $\mathbf{x}$ versus the sum of fixed bumps at each data point $\mathbf{x}_n$.



(a) Nonparametric RBF, small r (b) Nonparametric RBF, large r

(c) Parametric RBF, small r (b) Parametric RBF, large r

Figure 6.10: (a),(b) The nonparametric RBF; (c),(d) The parametric RBF. The toy data set has 3 points. The width r controls the smoothness of the hypothesis. The nonparametric RBF is step-like with large $|x|$ behavior determined by the target values at the peripheral data; the parametric RBF is bump-like with large x behavior that converges to zero.

Exercise 6.12

(a) For the Gaussian kernel, what is $g(\mathbf{x})$ as $\|\mathbf{x}\| \to \infty$ for the nonparametric RBF versus for the parametric RBF with fixed w_n?

(b) Let Z be the square feature matrix defined by $\mathrm{Z}_{nj} = \Phi_j(\mathbf{x}_n)$. Assume Z is invertible. Show that $g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{\Phi}(\mathbf{x})$, with $\mathbf{w} = \mathrm{Z}^{-1}\mathbf{y}$ exactly interpolates the data points. That is, $g(\mathbf{x}_n) = y_n$, giving $E_{\text{in}}(g) = 0$.

(c) Does the nonparametric RBF always have $E_{\text{in}} = 0$?

The parametric RBF has N parameters $w_1, \ldots, w_N$, ensuring we can always fit the data. When the data has stochastic or deterministic noise, this means we will overfit. The root of the problem is that we have too many parameters, one for each bump. Solution: restrict the number of bumps to $k \ll N$. If we restrict the number of bumps to k, then only k weights $w_1, \ldots, w_k$ need to be learned. Naturally, we will choose the weights that best fit the data.

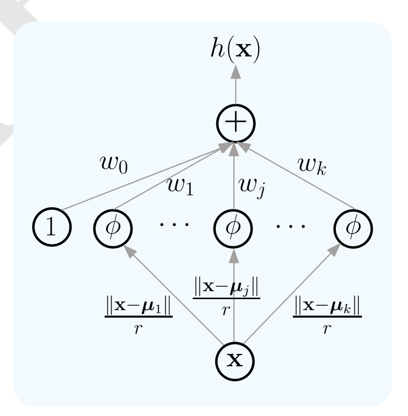
Where should the bumps be placed? Previously, with N bumps, this was

not an issue since we placed one bump on each data point. Though we cannot place a bump on each data point, we can still try to choose the bump centers so that they represent the data points as closely as possible. Denote the centers of the bumps by $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$. Then, a hypothesis has the parameterized form

$$h(\mathbf{x}) = w_0 + \sum_{j=1}^{k} w_j \phi\left(\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|}{r}\right) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\Phi}(\mathbf{x}), \tag{6.4}$$

where $\boldsymbol{\Phi}(\mathbf{x})^{\mathrm{T}} = [1, \Phi_1(\mathbf{x}), \ldots, \Phi_k(\mathbf{x})]$ and $\Phi_j(\mathbf{x}) = \phi(\|\mathbf{x} - \boldsymbol{\mu}_j\|/r)$. Observe that we have added back the bias term w_0.[11] For the nonparametric RBF or the parametric RBF with N bumps, we did not need the bias term. However, when you only have a small number of bumps and no bias term, the learning gets distorted if the y-values have non-zero mean (as is also the case if you do linear regression without the constant term).

The unknowns that one has to learn by fitting the data are the weights $w_0, w_1, \ldots, w_k$ and the bump centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ (which are vectors in $\mathbb{R}^d$). The model we have specified is called the *radial basis function network (RBF-network).* We can get the RBF-network model for classification by taking the sign of the output signal, and for logistic regression we pass the output signal through the sigmoid $\theta(s) = e^s/(1 + e^s)$.



A useful graphical representation of the model as a 'feed forward' network is illustrated. There are several important features that are worth discussing. First, the hypothesis set is very similar to a linear model except that the transformed features $\Phi_j(\mathbf{x})$ can depend on the data set (through the choice of the $\boldsymbol{\mu}_j$ which are chosen to fit the data). In the linear model of Chapter 3, the transformed features were fixed ahead of time. Because the $\boldsymbol{\mu}_j$ appear inside a nonlinear function, this is our first model that is not linear in its parameters. It is linear in the w_j, but nonlinear in the $\boldsymbol{\mu}_j$. It turns out that allowing the the basis functions to depend on the data adds significant power to this model over the linear model. We discuss this issue in more detail later, in Chapter 7.

Other than the parameters w_j and $\boldsymbol{\mu}_j$ which are chosen to fit the data, there are two high-level parameters k and r which specify the nature of the hypothesis set. These parameters control two aspects of the hypothesis set that were discussed in Chapter 5, namely the size of a hypothesis set (quantified by k, the number of bumps) and the complexity of a single hypothesis (quantified by r, which determines how 'wiggly' an individual hypothesis is).

[11] An alternative notation for the scale parameter $1/r$ is γ, and for the bias term w_0 is b (often used in the context of SVM, see Chapter 8).

It is important to choose a good value of k to avoid overfitting (too high a k) or underfitting (too low a k). A good strategy for choosing k is using cross validation. For a given k, a good simple heuristic for setting r is

$$r = \frac{R}{k^{1/d}},$$

where $R = \max_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|$ is the 'diameter' of the data. The rationale for this choice of r is that k spheres of radius r have a volume equal to $c_d k r^d = c_d R^d$ which is about the volume of the data (c_d is a constant depending only on d). So, the k bumps can 'cover' the data with this choice of r.

6.3.2 Fitting the Data

To complete our specification of the RBF-network model, we need to discuss how to fit the data. For a given k and r, to obtain our final hypothesis g, we need to determine the centers $\boldsymbol{\mu}_j$ and the weights w_j. In addition to the weights, we appear to have an abundance of parameters in the centers $\boldsymbol{\mu}_j$. Fortunately, we will determine the centers by choosing them to represent the inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$ without reference to the $y_1, \ldots, y_N$. The heavy lifting to fit the y_n will be done by the weights, so if there is overfitting, it is mostly due to the flexibility to choose the weights, not the centers.

For a given set of centers, $\boldsymbol{\mu}_j$, the hypothesis is linear in the w_j; we can therefore use our favorite algorithm from Chapter 3 to fit the weights *if the centers are fixed*. The algorithm is summarized as follows.

Fitting the RBF-network to the data (given k, r):

1: Using the inputs X, determine k centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
2: Compute the $N \times (k+1)$ feature matrix Z

$$\mathrm{Z}_{n,0} = 1 \qquad \mathrm{Z}_{n,j} = \Phi_j(\mathbf{x}_n) = \phi\left(\frac{\|\mathbf{x}_n - \boldsymbol{\mu}_j\|}{r}\right).$$

Each row of Z is the RBF-feature corresponding to $\mathbf{x}_n$ (where we have the dummy bias coordinate $\mathrm{Z}_{n,0} = 1$).
3: Fit the *linear* model $\mathrm{Z}\mathbf{w}$ to $\mathbf{y}$ to determine the weights $\mathbf{w}^*$.

We briefly elaborate on step 3 in the above algorithm.

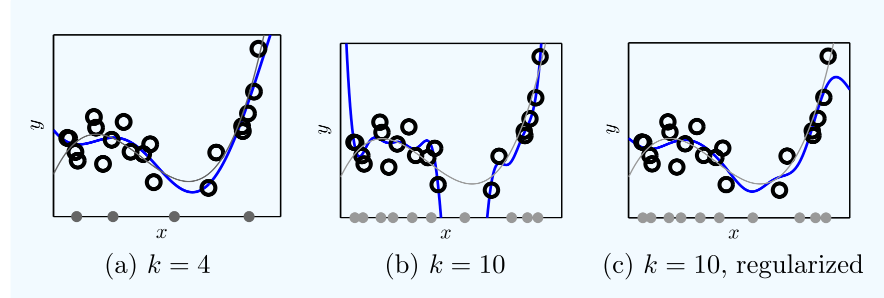
- For classification, you can use one of the algorithms for linear models from Chapter 3 to find a set of weights $\mathbf{w}$ that separate the positive points in Z from the negative points; the linear programming algorithm in Problem 3.6 would also work well.
- For regression, you can use the analytic pseudo-inverse algorithm

$$\mathbf{w}^* = \mathrm{Z}^\dagger \mathbf{y} = (\mathrm{Z}^{\mathrm{T}}\mathrm{Z})^{-1}\mathrm{Z}^{\mathrm{T}}\mathbf{y},$$

where the last expression is valid when Z has full column rank. Recall also the discussion about overfitting from Chapter 4. It is prudent to use regularization when there is stochastic or deterministic noise. With regularization parameter λ, the solution becomes $\mathbf{w}^* = (\mathrm{Z}^{\mathrm{T}}\mathrm{Z}+\lambda\mathrm{I})^{-1}\mathrm{Z}^{\mathrm{T}}\mathbf{y}$; the regularization parameter can be selected with cross validation.

- For logistic regression, you can obtain $\mathbf{w}^*$ by minimizing the logistic regression cross entropy error, using, for example, gradient descent. More details can be found in Chapter 3.

In all cases, the meat of the RBF-network is in determining the centers $\boldsymbol{\mu}_j$ and computing the radial basis feature matrix Z. From that point on, it is a straightforward linear model.



(a) $k = 4$ (b) $k = 10$ (c) $k = 10$, regularized

Figure 6.11: The parametric RBF-network using the same noisy data from Figure 6.7. (a) Small k. (b) Large k results in overfitting. (c) Large k with regularized linear fitting of weights ($\lambda = 0.05$). The centers μ_j are the gray shaded circles and $r = 1/k$. The target function is the light gray curve.

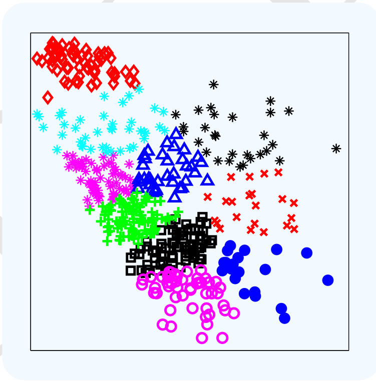
Figure 6.11 illustrates the RBF-network on the noisy data from Figure 6.7. We (arbitrarily) selected the centers μ_j by partitioning the data points into k equally sized groups and taking the average data point in each group as a center. When $k = 4$, one recovers a decent fit to f even though our centers were chosen somewhat arbitrarily. When $k = 10$, there is clearly overfitting, and regularization is called for.

Given the centers, fitting a RBF-network reduces to a linear problem, for which we have good algorithms. We now address this first step, namely how to determine these centers. Determining the optimal bump centers is the hard part, because this is where the model becomes essentially nonlinear. Luckily, the physical interpretation of this hypothesis set as a set of bumps centered on the $\boldsymbol{\mu}_j$ helps us. When there were N bumps (the nonparametric case), the natural choice was to place one bump on top of each data point. Now that there are only $k \ll N$ bumps, we should still choose the bump centers to closely represent the inputs in the data (we only need to match the $\mathbf{x}_n$, so we do not need to know the y_n). This is an *unsupervised* learning problem; in fact, a very important one.

One way to formulate the task is to require that no $\mathbf{x}_n$ be far away from a bump center. The $\mathbf{x}_n$ should cluster around the centers, with each center $\boldsymbol{\mu}_j$ representing one cluster. This is known as the *k-center problem*, known to be NP-hard. Nevertheless, we need a computationally efficient way to partition the data into k clusters and pick representative centers $\{\boldsymbol{\mu}_j\}_{j=1}^k$, one center for each cluster. One thing we learn from Figure 6.11 is that even for somewhat arbitrarily selected centers, the final result is still quite good, and so we don't have to find the absolutely optimal way to represent the data using k centers; a decent approximation will do. The k-means clustering algorithm is one way to pick a set of centers that are a decent representation of the data.

6.3.3 Unsupervised k-Means Clustering

Clustering is one of the classic unsupervised learning tasks. The k-means clustering algorithm is a simple yet powerful tool for obtaining clusters and cluster centers. We illustrate a clustering of 500 of the digits data into ten clusters. It is instructive to compare the clusters obtained here in an unsupervised manner with the classification regions of the k-NN rule in Figure 6.6(b). The clusters seem to mimic to some extent the classification regions, indicating that there is significant information in the input data alone.



We already saw the basic algorithm when we discussed partitioning for the branch and bound approach to finding the nearest neighbor. The goal of k-means clustering is to partition the input data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ into k sets $S_1, \ldots, S_k$ and select centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ for each cluster. The centers are representative of the data if every data point in cluster S_j is close to its corresponding center $\boldsymbol{\mu}_j$. For cluster S_j with center $\boldsymbol{\mu}_j$, define the squared error measure E_j to quantify the quality of the cluster,

$$E_j = \sum_{\mathbf{x}_n \in S_j} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2.$$

The error E_j measures how well the center $\boldsymbol{\mu}_j$ approximates the points in S_j. The k-means error function just sums this cluster error over all clusters,

$$E_{\text{in}}(S_1, \ldots, S_k; \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k) = \sum_{j=1}^{k} E_j = \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}(\mathbf{x}_n)\|^2, \qquad (6.5)$$

where $\boldsymbol{\mu}(\mathbf{x}_n)$ is the center of the cluster to which $\mathbf{x}_n$ belongs. We seek the partition $S_1, \ldots, S_k$ and centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ that minimize the k-means error.

Exercise 6.13

(a) Fix the clusters to $S_1, \ldots, S_k$. Show that the centers which minimize $E_{\text{in}}(S_1, \ldots, S_k; \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$ are the centroids of the clusters:

$$\boldsymbol{\mu}_j = \frac{1}{|S_j|} \sum_{\mathbf{x}_n \in S_j} \mathbf{x}_n.$$

(b) Fix the centers to $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$. Show that the clusters which minimize $E_{\text{in}}(S_1, \ldots, S_k; \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$ are obtained by placing into S_j all points for which the closest center is $\boldsymbol{\mu}_j$, breaking ties arbitrarily:

$$S_j = \{\mathbf{x}_n : \|\mathbf{x}_n - \boldsymbol{\mu}_j\| \leq \|\mathbf{x}_n - \boldsymbol{\mu}_\ell\| \text{ for } \ell = 1, \ldots, k\}.$$

Minimizing the k-means error is an NP-hard problem. However, the previous exercise says that if we fix a partition, then the optimal centers are easy to obtain. Similarly, if we fix the centers, then the optimal partition is easy to obtain. This suggests a very simple iterative algorithm which is known as *Lloyd's algorithm.* The algorithm starts with candidate centers and iteratively improves them until convergence.

Lloyd's Algorithm for k-Means Clustering:

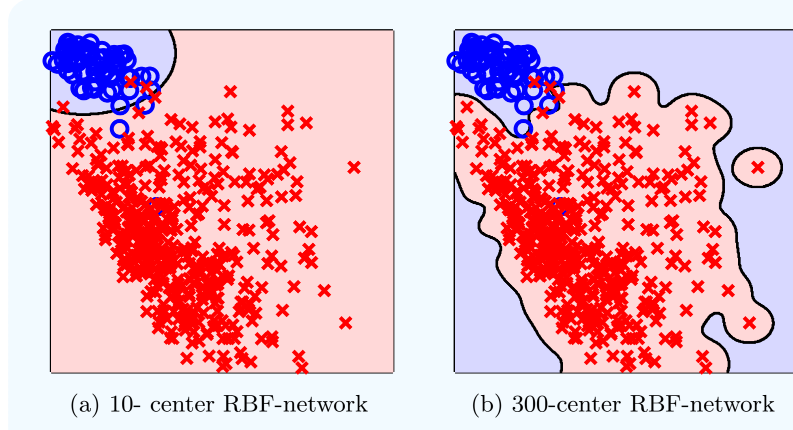
1: Initialize $\boldsymbol{\mu}_j$ (e.g. using the greedy approach on page 16).
2: Construct S_j to be all points closest to $\boldsymbol{\mu}_j$.
3: Update each $\boldsymbol{\mu}_j$ to equal the centroid of S_j.
4: Repeat steps 2 and 3 until E_{in} stops decreasing.

Lloyd's algorithm was the algorithm used to cluster the digits data into the ten clusters that were shown in the previous figure.

Exercise 6.14

Show that steps 2 and 3 in Lloyd's algorithm can never increase E_{in}, and hence that the algorithm must eventually stop iterating. *[Hint: There are only a finite number of different partitions]*

Lloyd's algorithm produces a partition which is 'locally optimal' in an unusual sense: given the centers, there is no better way to assign the points to clusters defined by the centers; and, given the partition, there is no better choice for the centers. However, the algorithm need not find the optimal partition as one might be able to improve the k-means error by simultaneously changing the centers and the cluster memberships. Lloyd's algorithm falls into a class of algorithms known as *E-M* (expectation-maximization) algorithms. It minimizes a complex error function by separating the variables to be optimized into two sets. If one set is known, then it is easy to optimize the other set, which is the basis for an iterative algorithm, such as with Lloyd's algorithm. It



(a) 10- center RBF-network (b) 300-center RBF-network

Figure 6.12: The RBF-network model for classifying the digits data ('1' versus 'not 1') with (a) 10 centers and (b) 300 centers. The centers are first obtained by the k-means clustering algorithm. The resulting classifier is obtained after fitting the weights using the linear regression algorithm for classification (see Chapter 3). The figures highlight the possibility of overfitting with too many centers.

is an on-going area of theoretical research to find algorithms which can guarantee near optimal clustering. There are some algorithms which achieve this kind of accuracy and run in $O(n2^{O(d)})$ time (curse of dimensionality). For our practical purposes, we do not need the best clustering *per se*; we just need a good clustering that is representative of the data, because we still have some flexibility to fit the data by choosing the weights w_j. Lloyd's algorithm works just fine in practice, and it is quite efficient. In Figure 6.12, we show the RBF-network classifier for predicting '1' versus 'not 1' when using 10 centers versus 300 centers. In both cases, the cluster centers were determined using Lloyd's algorithm. Just as with linear models, one can overfit with RBF-networks. To get the best classifier, one should use regularization and validation to combat overfitting.

Selecting the Number of Clusters k. We introduced clustering as a means to determine the centers of the RBF-network in supervised learning. The ultimate goal is out-of-sample prediction, so we need to choose k (the number of centers) to give enough flexibility without overfitting. In our supervised setting, we can minimize a cross validation error to choose k. However, more generally, clustering is a stand alone unsupervised task. With our digits data, a natural choice for k is ten, one cluster for each digit. Algorithms to choose k using only the unlabeled data without knowing the number of classes in the supervised task are also part of on-going research.

6.3.4 Justifications and Extensions of RBFs

We 'derived' RBFs as a natural extension to the k-nearest neighbor algorithm; a soft version of k-nearest neighbors, if you will. There are other ways to derive them. RBF-networks arise naturally through Tikhonov regularization for fitting nonlinear functions to the data – Tikhonov regularization penalizes curvature in the final hypothesis (see Problem 6.20). RBF-networks also arise naturally from noisy interpolation theory where one tries to achieve minimum expected in-sample error under the assumption that the $\mathbf{x}$-values are noisy; this is similar to regularization where one asks g to be nearly a constant equal to y_n in the neighborhood of $\mathbf{x}_n$.

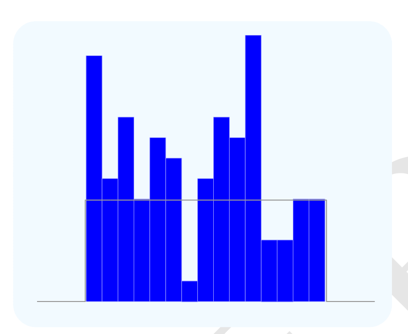
Our RBF-network had k identical bumps. A natural way to extend this model is to allow the bumps to have different shapes. This can be accomplished by choosing different scale parameters for each bump (so r_j instead of r). Further, the bumps need not be spherically symmetric. For the Gaussian kernel, the hypothesis for this more general RBF-network has the form:

$$h(\mathbf{x}) = \sum_{j=1}^{k} w_j e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^{\mathrm{T}}\Sigma_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)}. \tag{6.6}$$

The weights $\{w_j\}_{j=1}^k$, the centers $\{\boldsymbol{\mu}_j\}_{j=1}^k$ (vectors in $\mathbb{R}^d$) and the shapes $\{\Sigma_j\}_{j=1}^k$ ($d \times d$ positive definite symmetric covariance matrices) all need to be learned from the data. Intuitively, each bump j still represents a cluster of data centered at $\boldsymbol{\mu}_j$, but now the scale r is replaced by a 'scale matrix' Σ_j which captures the scale and correlations of points in the cluster. For the Gaussian kernel, the $\boldsymbol{\mu}_j$ are the cluster centroids, and the Σ_j are the cluster covariance matrices. We can fit the w_j using techniques for linear models, once the $\boldsymbol{\mu}_j$ and Σ_j are given. As with the simpler RBF-network, the location and shape parameters can be learned in an unsupervised way; an E-M algorithm similar to Lloyd's algorithm can be applied (Section 6.4.1 elaborates on the details). For Gaussian kernels, this unsupervised learning task is called learning a Gaussian Mixture Model (GMM).

6.4 Probability Density Estimation

Clustering was our first unsupervised method that tried to organize the input data. A cluster contains points that are similar to each other. The probability density of $\mathbf{x}$ is a generalization of clustering to a finer representation. Clusters can be thought of as regions of high probability. The basic task in probability density estimation is to estimate, for a given $\mathbf{x}$, how likely it is that you would generate inputs *similar* to $\mathbf{x}$. To answer this question we need to look at what fraction of the inputs in the data are similar to $\mathbf{x}$. Since similarity plays an important role, many of the similarity-based techniques in supervised learning have counterparts in the unsupervised task of probability density estimation. Vast tomes have been written on this topic, so we only skim its surface.



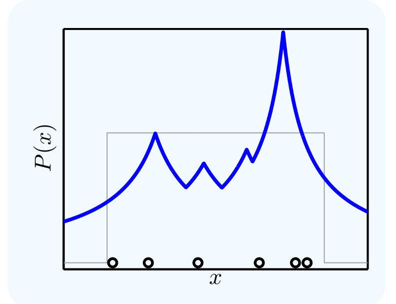
The Histogram. Given data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ generated independently from $P(\mathbf{x})$, the task is to learn the entire probability density P. Since $P(\mathbf{x})$ is the density of points at $\mathbf{x}$, the natural estimate is to use the in-sample density of points around $\mathbf{x}$. The *histogram* density estimate illustrated on the right uses a partition of $\mathcal{X}$ into uniform disjoint hypercubes (multidimensional intervals). Every point $\mathbf{x}$ falls into one bin and each bin has volume V. In one dimension, V is just the length of the interval. The estimate $\hat{P}(\mathbf{x})$ is the density of points in the hypercube that contains $\mathbf{x}$, normalized so that the integral is 1.

$$\hat{P}(\mathbf{x}) = \frac{1}{N} \cdot \frac{N(\mathbf{x})}{V},$$

where $N(\mathbf{x})$ is the number of data points in the hypercube containing $\mathbf{x}$. The density of points in this hypercube containing $\mathbf{x}$ is $N(\mathbf{x})/V$. The normalization by $1/N$ is to ensure that $\int d\mathbf{x}\ \hat{P}(\mathbf{x}) = 1$. The histogram estimate $\hat{P}(\mathbf{x})$ is not continuous. The main parameter which controls the quality of the estimate is V (or the number of bins, which is inversely related to V). Under mild assumptions, one recovers the true density as $V \to 0$ and $N \cdot V \to \infty$. This ensures that you are only using points very similar to $\mathbf{x}$ to estimate the density at $\mathbf{x}$ and there are enough such points being used. One choice is $V = 1/\sqrt{N}$.

Nearest Neighbor Density Estimation. Rather than using uniform hypercubes, one can use the distance to the kth nearest neighbor to determine the volume of the region containing $\mathbf{x}$. Let $r_{[k]}(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_{[k]}\|$ be the distance from $\mathbf{x}$ to its kth nearest neighbor, and $V_{[k]}(\mathbf{x})$ the volume of the spheroid centered at $\mathbf{x}$ of radius $r_{[k]}(\mathbf{x})$. In d-dimensions, $V_{[k]}$ and $r_{[k]}$ are related by

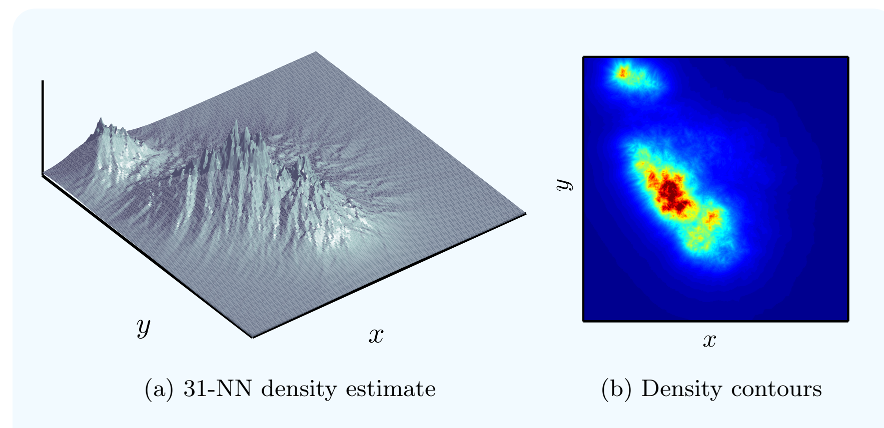
$$V_{[k]} = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} {r_{[k]}}^d.$$



The density of points in this spheroid is $k/V_{[k]}$, so we have the estimate

$$\hat{P}(\mathbf{x}) \quad = \quad c \cdot \frac{k}{V_{[k]}(\mathbf{x})},$$

where c is chosen by normalizing $\hat{P}$, so that $\int \mathrm{d}\mathbf{x} \quad \hat{P}(\mathbf{x}) = 1$. The 3-NN density estimate using 6 points from a uniform distribution is illustrated to the right. As is evident, the nearest neighbor density estimate is continuous but not smooth, having sharp spikes and troughs.



(a) 31-NN density estimate   (b) Density contours

Figure 6.13: The 31-NN density estimate for 500 digits data. (a) The probability density surface showing the spike behavior of the estimate. (b) Density contours (red is high, blue is low) from which we can visually identify the clusters in the data as regions of locally high density.

The nearest neighbor density estimate only works with a bounded input space because otherwise the estimate is not normalizable. In fact the tail of the nearest neighbor density estimate is decaying as $\|\mathbf{x}\|^{-d}$ which is fat-tailed. The nearest neighbor density estimate is nonparametric, and the usual convergence theorem holds. If $k \to \infty$ and $k/N \to 0$, then under mild assumptions on P, $\hat{P}$ converges to P (where the discrepancy between P and $\hat{P}$ is measured by the integrated squared error, $\int \mathrm{d}\mathbf{x}\ \ (P(\mathbf{x}) - \hat{P}(\mathbf{x}))^2$). Figure 6.13 illustrates the nearest neighbor density estimate using 500 randomly sampled digits data.

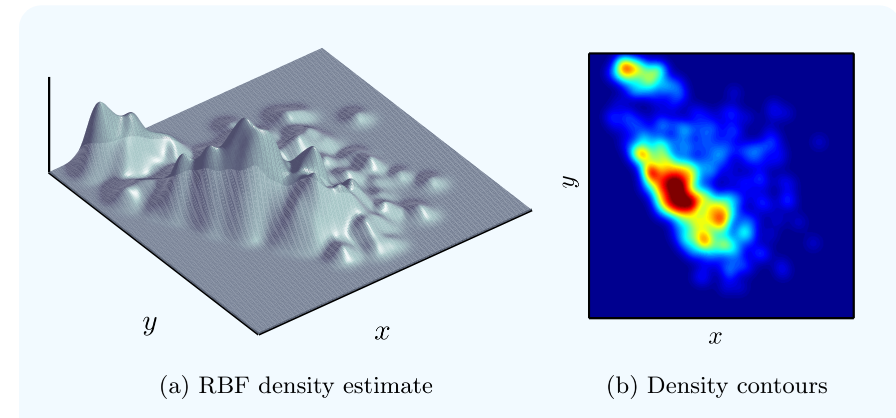
Parzen Windows: RBF Density Estimation. Perhaps the most common density estimation technique is the Parzen window. Recall that the kernel ϕ is a bump function which is positive. For density estimation, it is convenient to normalize ϕ so that $\int d\mathbf{x}\ \phi(\|\mathbf{x}\|) = 1$. The normalized Gaussian kernel is

$$\phi(z) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}z^2}.$$

One can verify that $\hat{P}(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ is a probability density, and for any $r > 0$,

$$\hat{P}(\mathbf{x}) = \frac{1}{r^d} \cdot \phi\left(\frac{\|\mathbf{x}\|}{r}\right)$$

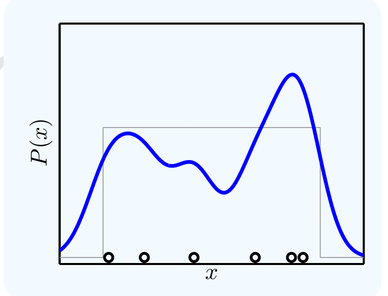
is also a density (r is the width of the bump). The Parzen window is similar to the nonparametric RBF in supervised learning: you have a bump with weight $\frac{1}{N}$ on each data point, and $\hat{P}(\mathbf{x})$ is a sum of the bumps:



(a) RBF density estimate (b) Density contours

Figure 6.14: Gaussian kernel Parzen window with $r = 0.02$ using 500 digits data. (a) The probability density surface is smooth but bumpy around the periphery of the data; the tail is exponential, dropping very quickly to zero away from the data. Compare this with the spiky more slowly decaying density produced by nearest neighbor in Figure 6.13. (b) Density contours (red is high, blue is low) highlighting the clusters in the data as regions of locally high density. Clusters are more readily visible with the smoother Parzen window than the nearest neighbor estimate.

$$\hat{P}(\mathbf{x}) = \frac{1}{Nr^d} \sum_{i=1}^{N} \phi\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{r}\right).$$



Since each bump integrates to 1, the scaling by $\frac{1}{N}$ ensures that $\hat{P}(\mathbf{x})$ integrates to 1. The figure to the right illustrates the Gaussian kernel Parzen window with six points from a uniform density. As opposed to the nearest neighbor density, the Parzen window is smooth, and has a thin tail. These properties are inherited from the Gaussian kernel, which is smooth and exponentially decaying. Figure 6.14 shows the Gaussian kernel Parzen window applied to our 500 randomly sampled digits data, with $r = 0.02$. Observe that the Parzen window can be bumpy around the periphery of the data. We can reduce this bumpiness by increasing r, which risks missing the finer structure at the heart of the data.

As with nearest neighbors, the Parzen window is nonparametric, modulo the choice of the kernel-width. If $r \to 0$ and $r^d N \to \infty$ as N grows, then under mild assumptions, $\hat{P}$ converges to P with respect to integrated square error. One choice for r is $1/\sqrt[2d]{N}$. One can also use cross validation to determine a good value of r that maximizes the likelihood of the validation set (recall that we used likelihood to derive the error function for logistic regression).

6.4.1 Gaussian Mixture Models (GMMs)

Just as the RBF-network is the parametric version of the nonparametric RBF, the Gaussian mixture model (GMM) is the parametric version of the Parzen window density estimate. The Parzen window estimate places a Gaussian bump at every data point; the GMM places just k bumps at centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$. The Gaussian kernel is the most commonly used and easiest to handle. In d-dimensions, the Gaussian density with center $\boldsymbol{\mu}$ and covariance matrix Σ is:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}.$$

Note that the center $\boldsymbol{\mu}$ is also the expected value, $\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}$; and the covariance between two features x_i, x_j is Σ_{ij}, $\mathbb{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}] = \Sigma$.

To derive the GMM, we use the following simple model of how the data is generated. There are k Gaussian distributions, with respective means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ and covariance matrices $\Sigma_1, \ldots, \Sigma_k$. To generate a data point $\mathbf{x}$, first pick a Gaussian $j \in \{1, \ldots, k\}$ according to probabilities $\{w_1, \ldots, w_k\}$, where $w_j > 0$ and $\sum_{j=1}^{k} w_j = 1$. After selecting Gaussian j, $\mathbf{x}$ is generated according to a Gaussian distribution with parameters $\boldsymbol{\mu}_j, \Sigma_j$. The probability density of $\mathbf{x}$ given that you picked Gaussian j is

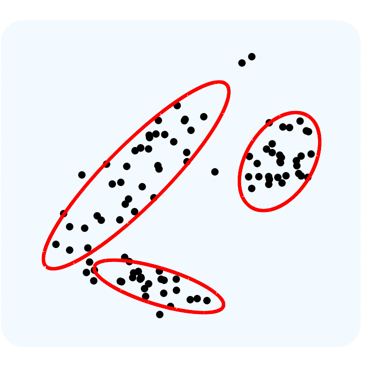
$$P(\mathbf{x}|j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \Sigma_j) = \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^{\mathrm{T}}\Sigma_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)}.$$

By the law of total probability, the unconditional probability density for $\mathbf{x}$ is

$$P(\mathbf{x}) = \sum_{j=1}^{k} P(\mathbf{x}|j)\,\mathbb{P}[j] = \sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \Sigma_j) \tag{6.7}$$

Compare (6.7) with (6.4). The GMM is just an RBF-network with weights w_j and a Gaussian kernel – a sum of Gaussian bumps, hence the name.

Each of the k Gaussian bumps represents a 'cluster' of the data. The probability density in Equation (6.7) puts a bump (Gaussian) of total probability (weight) w_j at the center $\boldsymbol{\mu}_j$; Σ_j determines the 'shape' of the bump. An example of some data generated from a GMM in two dimensions with $k = 3$ is shown on the right. The Gaussians are illustrated by a contour of constant probability. The different shapes of the clusters are controlled by the covariances Σ_j. Equation (6.7) is based on a *generative* model for the data, and the parameters of the model need to be learned from the actual data. These parameters are the mixture weights w_j, the centers $\boldsymbol{\mu}_j$ and the covariance matrices Σ_j.



To determine the unknown parameters in the GMM, we will minimize an in-sample error called the likelihood. We already saw the likelihood when we derived the in-sample error for logistic regression in Chapter 3. Among all the possible GMMs, each determined by a different choice for the parameters in Equation (6.7), we want to select one. Our criterion for choosing is that, for the chosen density estimate $\hat{P}$, the data should have a high probability of being generated. Since the data points are independent, the probability (density) for the data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ if the data were generated according to $\hat{P}(\mathbf{x})$ is

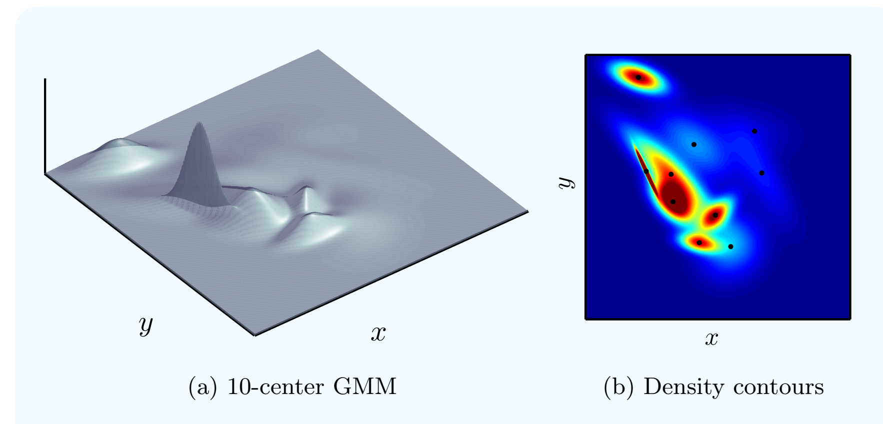
$$\hat{P}(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} \hat{P}(\mathbf{x}_n) = \prod_{n=1}^{N} \left(\sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j) \right).$$

This is the likelihood of a particular GMM specified by parameters $\{w_j, \boldsymbol{\mu}_j \Sigma_j\}$. The method of maximum likelihood selects the parameters which maximize $\hat{P}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$, or equivalently which minimize $-\ln \hat{P}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$. Thus, we may minimize the in-sample error:

$$\begin{aligned} E_{\text{in}}(w_j, \boldsymbol{\mu}_j, \Sigma_j) &= -\ln \hat{P}(\mathbf{x}_1, \ldots, \mathbf{x}_N) \\ &= -\sum_{i=1}^{N} \ln \left(\sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j) \right). \end{aligned} \tag{6.8}$$

To determine $\{w_j, \boldsymbol{\mu}_j, \Sigma_j\}$, we need an algorithm to minimize this in-sample error. Equation (6.8) is general in that we can replace the Gaussian density $\mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j)$ with any other parameterized density and we will obtain a non-Gaussian mixture model. Unfortunately, even for the friendly Gaussian mixture model, the summation inside the logarithm makes it very difficult to minimize the in-sample error, and we are going to need a heuristic. This heuristic is the Expectation Maximization (E-M) algorithm. The E-M algorithm is efficient and produces good results. To illustrate, we run a 10-center GMM on our sample of 500 digits data. The results are shown in Figure 6.15.

The Expectation Maximization (E-M) Algorithm The E-M algorithm was introduced in the late 1970s and is an important heuristic for maximizing the likelihood function. It is based on the notion of a *latent* (hidden, unmeasured, missing) piece of data that would make the optimization much easier. In the context of the Gaussian Mixture Model, suppose we knew which data points came from bump 1, bump 2, ..., bump k. The problem would suddenly become much easier, because we can estimate the center and covariance matrix of each bump using the data from that bump alone; further we can estimate the probabilities w_j by the fraction of data points in bump j. Unfortunately we do not know which data came from which bump, so we start with a guess, and iteratively improve this guess. The general algorithm is



(a) 10-center GMM (b) Density contours

Figure 6.15: The 10-center GMM for 500 digits data. (a) The probability density surface. The bumps are no longer spherical as in the Parzen window. The bump shapes are determined by the covariance matrices. (b) Density contours (red is high, blue is low) with the centers as black dots. The centers identify the 'clusters' (compare with the k-means clusters on page 31).

E-M Algorithm for GMMs:

1: Start with estimates for the bump membership of each $\mathbf{x}_n$.
2: Estimates of $w_j, \boldsymbol{\mu}_j, \Sigma_j$ given the bump memberships.
3: Update the bump memberships given $w_j, \boldsymbol{\mu}_j, \Sigma_j$; iterate to step 2 until convergence.

To mathematically specify the algorithm, we will add one more ingredient, namely that the bump memberships need not be all or nothing. Specifically, at iteration t, let $\gamma_{nj}(t) \geq 0$ be the 'fraction' of data point $\mathbf{x}_n$ that belongs to bump j, with $\sum_{j=1}^{k} \gamma_{nj} = 1$ (the entire point is allocated among all the bumps); you can view γ_{nj} as the probability that $\mathbf{x}_n$ was generated by bump j. The 'number' of data points belonging to bump j is given by

$$N_j = \sum_{n=1}^{N} \gamma_{nj}.$$

The γ_{nj} are the hidden variables that we do not know, but if we did know the γ_{nj}, then we could compute estimates of $w_j, \boldsymbol{\mu}_j, \Sigma_j$:

$$\begin{aligned} w_j &= \frac{N_j}{N}; \\ \boldsymbol{\mu}_j &= \frac{1}{N_j} \sum_{n=1}^{N} \gamma_{nj} \mathbf{x}_n; \\ \Sigma_j &= \frac{1}{N_j} \sum_{n=1}^{N} \gamma_{nj} \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} - \boldsymbol{\mu}_j \boldsymbol{\mu}_j^{\mathrm{T}}. \end{aligned} \tag{6.9}$$

Intuitively, the weights are the fraction of data belonging to bump j; the means are the average data point belonging to bump j where we take into account that only a fraction of a data point may belong to a bump; and, the covariance matrix is the weighted covariance of the data belonging to the bump. Once we have these new estimates of the parameters, we can update the bump memberships γ_{nj}. To get $\gamma_{nj}(t+1)$, we compute the probability that data point $\mathbf{x}_n$ came from bump j *given* the parameters $w_j, \boldsymbol{\mu}_j, \Sigma_j$. We want

$$\gamma_{nj}(t+1) = \mathbb{P}[j|\mathbf{x}_n].$$

By an application of Bayes rule,

$$\mathbb{P}[j|\mathbf{x}_n] = \frac{P(\mathbf{x}_n|j)\,\mathbb{P}\,[j]}{P(\mathbf{x}_n)} = \frac{\mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j) \cdot w_j}{P(\mathbf{x}_n)}.$$

We won't have to compute $P(\mathbf{x}_n)$ in the denominator because it is independent of j and can be fixed by the normalization condition $\sum_{j=1}^{k} \gamma_{nj} = 1$. We thus arrive at the update for the bump memberships,

$$\gamma_{nj}(t+1) = \frac{w_j \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \Sigma_j)}{\sum_{\ell=1}^{k} w_\ell \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_\ell, \Sigma_\ell)}.$$

After updating the bump memberships, we iterate back to the parameters and continue until convergence. To complete the specification of the algorithm, we need to specify the initial bump memberships. A simple heuristic is to use the k-means clustering algorithm described earlier to obtain a 'hard' partition with $\gamma_{nj} \in \{0, 1\}$, and proceed from there.

Exercise 6.15

What would happen in the E-M algorithm described above if you initialized the bump memberships uniformly to $\gamma_{nj} = 1/k$?

The E-M algorithm is a remarkable example of a learning algorithm that 'bootstraps' itself to a solution. The algorithm starts by guessing some values for unknown quantities that you would like to know. The guess is probably quite wrong, but nevertheless the algorithm makes inferences based on the incorrect guess. These inferences are used to slightly improve the guess. The guess and inferences slightly improve each other in this way until at the end you have bootstrapped yourself to a decent guess at the unknowns, as well as a good inference based on those unknowns.

6.5 Problems

Problem 6.1 Consider the following data set with 7 data points.

$$\left(\begin{bmatrix}1\\0\end{bmatrix}, -1\right)\left(\begin{bmatrix}0\\1\end{bmatrix}, -1\right)\left(\begin{bmatrix}0\\-1\end{bmatrix}, -1\right)\left(\begin{bmatrix}-1\\0\end{bmatrix}, -1\right)$$
$$\left(\begin{bmatrix}0\\2\end{bmatrix}, +1\right)\left(\begin{bmatrix}0\\-2\end{bmatrix}, +1\right)\left(\begin{bmatrix}-2\\0\end{bmatrix}, +1\right)$$

(a) Show the decision regions for the 1-NN and 3-NN rules.

(b) Consider the non-linear transform

$$\begin{bmatrix}x_1\\x_2\end{bmatrix} \mapsto \begin{bmatrix}z_1\\z_2\end{bmatrix} = \begin{bmatrix}\sqrt{x_1^2+x_2^2}\\ \arctan(x_2/x_1)\end{bmatrix},$$

which maps $\mathbf{x}$ to $\mathbf{z}$. Show the classification regions in the $\mathbf{x}$-space for the 1-NN and 3-NN rules implemented on the data in the $\mathbf{z}$-space.

Problem 6.2 Use the same data from the previous problem.

(a) Let the mean of all the -1 points be $\boldsymbol{\mu}_{-1}$ and the mean of all the $+1$ points be $\boldsymbol{\mu}_{+1}$. Suppose the data set were condensed into the two *prototypes* $\{(\boldsymbol{\mu}_{-1}, -1),\ (\boldsymbol{\mu}_{+1}, +1)\}$ (these points need not be data points, so they are called prototypes). Plot the classification regions for the 1-NN rule using the condensed data. What is the in-sample error?

(b) Consider the following approach to condensing the data. At each step, merge the two closest points of the same class as follows:

$$(\mathbf{x}, c) + (\mathbf{x}', c) \to (\tfrac{1}{2}(\mathbf{x}+\mathbf{x}'), c).$$

Again, this method of condensing produces prototypes. Continue condensing until you have two points remaining (of different classes).
Plot the 1-NN rule with the condensed data. What is the in-sample error?

Problem 6.3 Show that the k-nearest neighbor rule with distance defined by $d(\mathbf{x}, \mathbf{x}') = (\mathbf{x}-\mathbf{x}')^{\mathrm{T}}\mathrm{Q}(\mathbf{x}-\mathbf{x}')$, where Q is positive semi-definite, is equivalent to the k-nearest neighbor rule with the standard Euclidean distance in some transformed feature space. Explicitly construct this space. What is the dimension of this space. *[Hint: Think about the rank of* Q*.]*

Problem 6.4 For the double semi-circle problem in Problem 3.1, plot the decision regions for the 1-NN and 3-NN rules.

Problem 6.5 Show that each of the Voronoi regions in the Voronoi diagram for any data set is convex. (A set $\mathcal{C}$ is convex if for any $\mathbf{x}, \mathbf{x}' \in \mathcal{C}$ and any $\lambda \in [0, 1]$, $\lambda\mathbf{x} + (1 - \lambda)\mathbf{x}' \in \mathcal{C}$.)

Problem 6.6 For linear regression with weight decay, $g(\mathbf{x}) = \mathbf{x}^{\text{T}}\mathbf{w}_{\text{reg}}$. Show that

$$g(\mathbf{x}) = \sum_{n=1}^{N} \mathbf{x}^{\text{T}}(\mathrm{Z}^{\text{T}}\mathrm{Z} + \lambda\Gamma^{\text{T}}\Gamma)^{-1}\mathbf{x}_n y_n.$$

A *kernel representation* of a hypothesis g is a representation of the form

$$g(\mathbf{x}) = \sum_{n=1}^{N} K(\mathbf{x}, \mathbf{x}_n) y_n,$$

where $K(\mathbf{x}, \mathbf{x}')$ is the kernel function. What is the kernel function in this case?

One can interpret the kernel representation of the final hypothesis from linear regression as a similarity method, where g is a weighted sum of the target values $\{y_n\}$, weighted by the "similarity" $K(\mathbf{x}, \mathbf{x}_n)$ between the point $\mathbf{x}$ and the data point $\mathbf{x}_n$. Does this look similar ☺ to RBFs?

Problem 6.7 Consider the hypothesis set $\mathcal{H}$ which contains all labeled Voronoi tessellations on K points. Show that $d_{\text{VC}}(\mathcal{H}) = K$.

Problem 6.8 Suppose the target function is deterministic, so $\pi(\mathbf{x}) = 0$ or $\pi(\mathbf{x}) = 1$. The decision boundary implemented by f is defined as follows. The point $\mathbf{x}$ is on the decision boundary if every ball of positive radius centered on $\mathbf{x}$ contains both positive and negative points. Conversely if $\mathbf{x}$ is not on the decision boundary, then some ball around $\mathbf{x}$ contains only points of one classification.

Suppose the decision boundary (a set of points) has probability zero. Show that the simple nearest neighbor rule will asymptotically (in N) converge to optimal error E^*_{out} (with probability 1).

Problem 6.9 Assume that the support of P is the unit cube in d dimensions. Show that for any $\epsilon, \delta > 0$, there is a sufficiently large N for which, with probability at least $1 - \delta$,

$$\sup_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - \mathbf{x}_{(1)}(\mathbf{x})\| \leq \epsilon.$$

[Hint: Partition $[0, 1]^d$ into disjoint hypercubes of size $\epsilon/\sqrt{d}$. For any $\mathbf{x} \in [0, 1]^d$, show that at least one of these hypercubes is completely contained in

$B_\epsilon(\mathbf{x})$. Let $p > 0$ be the minimum P-measure of a hypercube. Since the number of hypercubes is finite, use a union bound to show that with high probability, every hypercube will contain at least k points for any fixed k as $N \to \infty$. Conclude that $\|\mathbf{x} - \mathbf{x}_{(k)}\| \le \epsilon$ with high probability as $N \to \infty$.]

Problem 6.10 Let $E_{\text{out}}(k) = \lim_{N\to\infty} E_{\text{out}}(g_N(k))$, where $g_N(k)$ is the k-nearest neighbor rule on a data set of size N, where k is odd. Let E^*_{out} is the optimal out-of-sample probability of error. Show that (with probability 1),

$$E^*_{\text{out}} \le E_{\text{out}}(k) \le E_{\text{out}}(k-2) \le \cdots \le E_{\text{out}}(1) \le 2E^*_{\text{out}}.$$

Problem 6.11 For the 1-NN rule in two dimensions ($d = 2$) and data set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, consider the Voronoi diagram and let V_n be the Voronoi region containing the point $\mathbf{x}_n$. Two Voronoi regions are adjacent if they have a face in common (in this case an edge). Mark a point $\mathbf{x}_n$ if the classification of every Voronoi region neighboring V_n is the same as y_n. Now condense the data by removing all marked points.

(a) Show that the condensed data is consistent with the full data for the 1-NN rule.

(b) How does the out-of-sample error for the 1-NN rule using condensed data compare with the 1-NN rule on the full data (worst case and on average)?

Problem 6.12 For the 1-NN rule, define the *influence set* S_n of point $\mathbf{x}_n$ as the set of points of the same class as $\mathbf{x}_n$ which are closer to $\mathbf{x}_n$ than to a point of a different class. So $\mathbf{x}_m \in S_n$ if

$$\|\mathbf{x}_n - \mathbf{x}_m\| < \|\mathbf{x}_{m'} - \mathbf{x}_m\| \text{ for all } m' \text{ with } y_{m'} \ne y_m.$$

We do not allow $\mathbf{x}_n$ to be in S_n. Suppose the largest influence set is S_{n^*}, the influence set of $\mathbf{x}_{n^*}$ (break ties according to the data point index). Remove all points in S_{n^*}. Update the remaining influence sets by deleting $\mathbf{x}_{n^*}$ *and* S_{n^*} from them. Repeat this process until all the influence sets are empty. The set of points remaining is a condensed subset of $\mathcal{D}$.

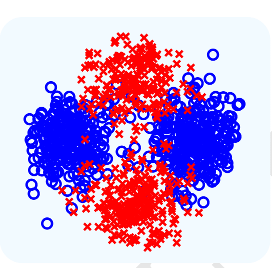
(a) Show that the remaining condensed set is training set consistent.

(b) Do you think this will give a smaller or larger condensed set than the CNN algorithm discussed in the text. Explain your intuition?

(c) Give an efficient algorithm to implement this greedy condensing. What is your run time.

Problem 6.13 Construct a data set with 1,000 points as shown.

The data are generated from a 4-center GMM. The centers are equally spaced on the unit circle, and covariance matrices all equal to $\sigma\mathrm{I}$ with $\sigma = 0.15$. The weight of each Gaussian bump is $\frac{1}{4}$. Points generated from the first and third centers have $y_n = +1$ and the other two centers are -1. To generate a point, first determine a bump (each has probability $\frac{1}{4}$) and then generate a point from the Gaussian for that bump.



(a) Implement the CNN algorithm in the text and the condensing algorithm from the previous problem.

(b) Run your algorithm on your generated data and give a plot of the condensed data in each case.

(c) Repeat this experiment 1,000 times and compute the average sizes of the condensed sets for the two algorithms.

Problem 6.14 Run the condensed nearest neighbor (CNN) algorithm for 3-NN on the digits data. Use all the data for classifying "1" versus "not 1".

(a) Set $N = 500$ and randomly split your data into a training set of size N and use the remaining data as a test/validation set.

(b) Use the 3-NN algorithm with all the training data and evaluate its performance: report E_{in} and E_{test}.

(c) Use the CNN algorithm to condense the data. Evaluate the performance of the 3-NN rule with the condensed data: report E_{in} and E_{out}.

(d) Repeat parts (b) and (c) using 1,000 random training-test splits and report the average E_{in} and E_{out} for the full versus the condensed data.

Problem 6.15 This problem asks you to perform an analysis of the branch and bound algorithm in an idealized setting. Assume the data is partitioned and each cluster with two or more points is partitioned into two sub-clusters of exactly equal size. (The number of data points is a power of 2).

When you run the branch and bound algorithm for a particular test point $\mathbf{x}$, sometimes the bound condition will hold, and sometimes not. If you generated the test point $\mathbf{x}$ randomly, the bound condition will hold with some probability. Assume the bound condition will hold with probability at least $p \geq 0$ at *every* branch, independently of what happened at other branches.

Let $T(N)$ be the expected time to find the nearest neighbor in a cluster with N points. Show: $T(N) = O(d \log N + dN^{\log_2(2-p)})$ (sublinear for $p > 0$).

[Hint: Let $N = 2^k$; show that $T(N) \leq 2d + T(\frac{N}{2}) + (1-p)T(\frac{N}{2})$.]

Problem 6.16

(a) Generate a data set of 10,000 data points uniformly in the unit square $[0, 1]^2$ to test the performance of the branch and bound method:

(i) Construct a 10-partition for the data using the simple greedy heuristic described in the text.

(ii) Generate 10,000 random query points and compare the running time of obtaining the nearest neighbor using the partition with branch and bound versus the brute force approach which does not use the partition.

(b) Repeat (a) but instead generate the data from a mixture of 10 gaussians with centers randomly distributed in $[0, 1]^2$ and identical covariances for each bump equal to $\sigma\mathrm{I}$ where $\sigma = 0.1$.

(c) Explain your observations.

(d) Does your decision to use the branch and bound technique depend on how many test points you will need to evaluate?

Problem 6.17 Using Exercise 6.8, assume that $p = \frac{1}{2}$ (the probability that the bound condition will hold at any branch point). Estimate the asymptotic running time to estimate the out-of-sample error with 10-fold cross validation to select the value of k, and compare with Exercise 6.6.

Problem 6.18 An alternative to the k-nearest neighbor rule is the r-nearest neighbor rule: classify a test point $\mathbf{x}$ using the majority class among all neighbors $\mathbf{x}_n$ within distance r of $\mathbf{x}$. The r-nearest neighbor explicitly enforces that all neighbors contributing to the decision must be close; however, it does not mean that there will be many such neighbors. Assume that the support of $P(\mathbf{x})$ is a compact set.

(a) Show that the expected number of neighbors contributing to the decision for any particular $\mathbf{x}$ is order of Nr^d.

(b) Argue that as N grows, if $Nr^d \to \infty$ and $r \to 0$, then the classifier approaches optimal.

(c) Give one example of such a choice for r (as a function of N, d).

Problem 6.19 For the full RBFN in Equation (6.6) give the details of a 2-stage algorithm to fit the model to the data. The first stage determines the parameters of the bumps (their centers and covariance matrices); and, the second stage determines the weights. *[Hint: For the first stage, think about the E-M algorithm to learn a Gaussian mixture model.]*

Problem 6.20 **[RBFs from regularization]** This problem requires advanced calculus. Let $d = 1$; we wish to minimize a regularized error

$$E_{\text{aug}}(h) = \sum_{i=1}^{N}(h(x_i) - y_i)^2 + \lambda \sum_{k=0}^{\infty} a_k \int_{-\infty}^{\infty} dx \ \left(h^{(k)}(x)\right)^2,$$

where λ is the regularization parameter. Assume $h^{(k)}(x)$ (the kth derivative of h) decays to to 0 as $|x| \to \infty$. Let $\delta(x)$ be the Dirac delta function.

(a) How would you select a_k to penalize how curvy or wiggly h is?

(b) [Calculus of Variations] Show that

$$E_{\text{aug}}(h) = \int_{-\infty}^{\infty} dx \ \left[\sum_{i=1}^{N}(h(x) - y_i)^2\delta(x - x_i) + \lambda \sum_{k=0}^{\infty} a_k \left(h^{(k)}(x)\right)^2\right].$$

Now find a stationary point of this functional: perturb h by δh and assume that for all k, $\delta h^{(k)} \to 0$ as $|x| \to 0$. Compute the change $\delta E_{\text{aug}}(h) = E_{\text{aug}}(h + \delta h) - E_{\text{aug}}(h)$ and set it to 0. After integrating by parts and discarding all terms of higher order than linear in δh, and setting all boundary terms from the integration by parts to 0, derive the following condition for h to be stationary (since δh is arbitrary),

$$\sum_{i=1}^{N}(h(x) - y_i)\delta(x - x_i) + \lambda \sum_{k=0}^{\infty}(-1)^k a_k h^{(2k)}(x) = 0.$$

(c) [Green's Functions] $\text{L} = \sum_{k=0}^{\infty}(-1)^k a_k \frac{d^k}{dx^k}$ is a linear differential operator. The Green's function $G(x, x')$ for L satisfies $\text{L}G(x, x') = \delta(x - x')$. Show that we can satisfy the stationarity condition by choosing

$$h(x) = \sum_{i=1}^{N} w_i G(x, x_i),$$

with $\mathbf{w} = (\text{G} + \lambda\text{I})^{-1}\mathbf{y}$, where $\text{G}_{ij} = G(x_i, x_j)$. ($h$ resembles an RBF with the Green's function as kernel. If L is translation and rotation invariant then $G(\mathbf{x}, \mathbf{x}') = G(\|\mathbf{x} - \mathbf{x}'\|)$, and we have an RBF.)

(d) [Computing the Green's Function] Solve $\text{L}G(x, x') = \delta(x - x')$ to get G. Define the Fourier transform and its inverse,

$$\hat{G}(f, x') = \int_{-\infty}^{\infty} dx \, e^{2\pi i f x} G(x, x'), \qquad \hat{G}(x, x') = \int_{-\infty}^{\infty} df \, e^{-2\pi i f x} \hat{G}(f, x').$$

Fourier transform both sides of $\text{L}G(x, x') = \delta(x - x')$, integrate by parts, assuming that the boundary terms vanish, and show that

$$G(x, x') = \int_{-\infty}^{\infty} df \frac{e^{2\pi i f(x' - x)}}{Q(f)},$$

where $Q(f) = \sum_{k=0}^{\infty} a_k (2\pi f)^{2k}$ is an even function whose power series expansion is determined by the a_k. If $a_k = 1/(2^k k!)$, what is $Q(f)$. Show that in this case the Green's function is the Gaussian kernel,

$$G(x, x') = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-x')^2}.$$

(For regularization that penalizes a particular combination of the derivatives of h, the *optimal* non-parametric regularized fit is a Gaussian kernel RBF.) *[Hint: You may need: $\int_{-\infty}^{\infty} dt\ e^{-at^2+bt} = \sqrt{\frac{\pi}{a}} e^{b^2/4a}$, $Re(a) > 0$.]*

Problem 6.21 Develop a linear programming approach to classification with similarity oracle $d(\cdot, \cdot)$ (as in Problem 3.6). Assume RBF-like hypotheses:

$$h(\mathbf{x}) = \text{sign}\left(w_0 + \sum_{i=1}^{N} w_i d(\mathbf{x}, \mathbf{x}_i) \right),$$

where $\mathbf{w}$ is the weight parameter to be determined by fitting the data. Pick the weights that fit the data and minimize the sum of weight sizes $\sum_{i=1}^{N} |w_i|$ (*lasso* regularization where we don't penalize w_0).

(a) Show that to find the weights, one solves the minimization problem:

$$\underset{\mathbf{w}}{\text{minimize}} \sum_{i=0}^{N} |w_i| \quad \text{s.t. } y_n \left(w_0 + \sum_{i=1}^{N} w_i d(\mathbf{x}_n, \mathbf{x}_i) \right) \geq 1.$$

Do you expect overfitting?

(b) Suppose we allow some error in the separation, then

$$y_n \left(w_0 + \sum_{i=1}^{N} w_i d(\mathbf{x}_n, \mathbf{x}_i) \right) \geq 1 - \zeta_n,$$

where $\zeta_n \geq 0$ are slack variables that measure the degree to which the data point $(\mathbf{x}_n, yn)$ has been misclassified. The total error is $\sum_{n=1}^{N} \zeta_n$. If you minimize a combination of the total weight sizes and the error with emphasis C on error, then argue that the optimization problem becomes

$$\begin{aligned} \underset{\mathbf{w},\boldsymbol{\zeta}}{\text{minimize}} \quad & \sum_{n=1}^{N} |w_n| + C \sum_{n=1}^{N} \zeta_n, \\ \text{s.t.} \quad & y_n \left(w_0 + \sum_{i=1}^{N} w_n d(\mathbf{x}_n, \mathbf{x}_i) \right) \geq 1 - \zeta_n, \\ & \zeta_n \geq 0, \end{aligned} \tag{6.10}$$

where the inequalities must hold for $n = 1, \ldots, N$.

The minimization trades off sparsity of the weight vector with the extent of misclassification. To encourage smaller in-sample error, one sets C to be large.

Problem 6.22 Show that the minimization in (6.10) is a linear program:

$$\begin{aligned}
\underset{\mathbf{w},\boldsymbol{\zeta},\boldsymbol{\alpha}}{\text{minimize}} \quad & \sum_{n=1}^{N} \alpha_n + C \sum_{n=1}^{N} \zeta_n, \\
\text{s.t.} \quad & -\alpha_n \le w_n \le \alpha_n, \\
& y_n \left(w_0 + \sum_{i=1}^{N} w_n d(\mathbf{x}_n, \mathbf{x}_i) \right) \ge 1 - \zeta_n, \\
& \zeta_n \ge 0,
\end{aligned}$$

where the inequalities must hold for $n = 1, \ldots, N$. Formulate this linear program in a standard form as in Problem 3.6. You need to specify what the parameters $\mathrm{A}, \mathbf{a}, \mathbf{b}$ are and what the optimization variable $\mathbf{z}$ is.

[Hint: Use auxiliary variables $\alpha_1, \ldots, \alpha_N$ to rewrite $|w_n|$ using linear functions.]

Problem 6.23 Consider a data distribution, $P(\mathbf{x}, y)$ which is a mixture of k Gaussian distributions with means $\{\boldsymbol{\mu}_j\}_{j=1}^{k}$ and covariance matrices $\{\Sigma_j\}_{j=1}^{k}$; each Gaussian has probability $p_j > 0$ of being selected, $\sum_{j=1}^{k} p_j = 1$; each Gaussian generates a positive label with probability π_j. To generate $(\mathbf{x}, y)$, first select a Gaussians using probabilities $p_1, \ldots, p_k$. If Gaussian ℓ is selected, generate $\mathbf{x}$ from this Gaussian distribution, with mean $\boldsymbol{\mu}_\ell$ and covariance Σ_ℓ, and $y = +1$ with probability π_ℓ ($y = -1$ otherwise).

For test point $\mathbf{x}$, show that the classifier with minimum error probability is

$$f(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^{k} w_j e^{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^{\mathrm{T}} \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)} \right),$$

where $w_j = p_j(2\pi_j - 1)$. *[Hint: Show that the optimal decision rule can be written $f(\mathbf{x}) = sign(\mathbb{P}[+1|\mathbf{x}] - \mathbb{P}[-1|\mathbf{x}])$. Use Bayes' theorem and simplify.]*

(This is the RBF-network for classification. Since $\boldsymbol{\mu}_j, \Sigma_j, p_j, \pi_j$ are unknown, they must be fit to the data. This problem shows the connection between the RBF-network for classification and a very simple probabilistic model of the data. The Bayesians often view the RBF-network through this lens.)

Problem 6.24 **[Determining the Number of Clusters k]** Use the same input data ($\mathbf{x}_n$) as in Problem 6.13.

(a) Run Lloyd's k-means clustering algorithm, for $k = 1, 2, 3, 4, \ldots$, outputting the value of the k-means objective $E_{\text{in}}(k)$.

(b) Generate benchmark random data of the same size, uniformly over the smallest axis-aligned square containing the the actual data (this data has no well defined clusters). Run Lloyd's algorithm on this random data for $k = 1, 2, 3, 4, \ldots$. Repeat for several such random data sets to obtain the average k-means error as a function of k, $E_{\text{in}}^{\text{rand}}(k)$.

(c) Compute and plot the *gap* statistic (as a function of k)

$$G(k) = \log E_{\text{in}}^{\text{rand}}(k) - \log E_{\text{in}}(k).$$

(d) Argue that the maximum of the gap statistic is a reasonable choice for the number of clusters to use. What value for k do you get?

(e) Repeat with different choices for σ in Problem 6.13, and plot the value of k chosen by the gap statistic versus σ. Explain your result.

Problem 6.25 **[Novelty Detection]** Novelty corresponds to the arrival of a new cluster. Use the same input data ($\mathbf{x}_n$) as in Problem 6.13.

(a) Use the method of Problem 6.24 to determine the number of clusters.

(b) Add data (one by one) from a 5th Gaussian centered on the origin with the same covariance matrix as the other four Gaussians. For each new point, recompute the number of clusters using the method of Problem 6.24. Plot the number of clusters versus ℓ the amount of data added.

(c) Repeat (b) and plot, as a function of ℓ, the average increase in the number of clusters from adding ℓ data points.

(d) From your plot in (c), estimate ℓ^*, the number of data points needed to identify the existence of this new cluster.

(e) Vary σ (the variance parameter of the Gaussians) and plot ℓ^* versus σ. Explain your results.

Problem 6.26 Let $V = \{v_1, \ldots, v_M\}$ be a universe of objects. Define the distance between two sets $S_1, S_2 \subseteq V$ by

$$d(S_1, S_2) = 1 - J(S_1, S_2),$$

where $J(S_1, S_2) = |S_1 \cap S_2|/|S_1 \cup S_2|$ is the Jaccard coefficient. Show that $d(\cdot, \cdot)$ is a metric satisfying non-negativity, symmetry and the triangle inequality.

Problem 6.27 Consider maximum likelihood estimation of the parameters of a GMM in one dimension from data $x_1, \ldots, x_N$. The probability density is

$$P(x) = \sum_{k=1}^{K} \frac{w_k}{2\pi\sigma_k^2} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right).$$

(a) If $K = 1$ what are the maximum likelihood estimate of μ_1 and σ_1^2.

(b) If $K > 1$, show that the maximum likelihood estimate is not well defined; specifically that the maximum of the likelihood function is infinite.

(c) How does the E-M algorithm perform? Under what conditions does the E-M algorithm converge to a well defined estimator?

(d) To address the problem in (b), define the "regularized" likelihood: For each x_n, define the 2ϵ-interval $B_n = [x_n - \epsilon, x_n + \epsilon]$. The ϵ-regularized likelihood is the probability that each $x_n \in B_n$. Intuitively, one does not measure x_n, but rather a 2ϵ-interval in which x_n lies.

(i) Show that

$$\mathbb{P}[x_n \in B_n] = \sum_{k=1}^{K} w_k \left(F_{\mathcal{N}}\left(\frac{x_i + \epsilon - \mu_k}{\sigma_k}\right) - F_{\mathcal{N}}\left(\frac{x_i + \epsilon - \mu_k}{\sigma_k}\right)\right),$$

where $F_{\mathcal{N}}(\cdot)$ is the standard normal distribution function.

(ii) For fixed ϵ, show that the maximum likelihood estimator is now well defined. What about in the limit as $\epsilon \to 0$?

(iii) Give an E-M algorithm to maximize the ϵ-regularized likelihood.

Problem 6.28 Probability density estimation is a very general task in that supervised learning can be posed as probability density estimation. In supervised learning, the task is to learn $f(\mathbf{x})$ from $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$.

(a) Let $P(\mathbf{x}, y)$ be the joint distribution of $(\mathbf{x}, y)$. For the squared error, $E_{\text{out}}(h) = \mathbb{E}_{P(\mathbf{x},y)}[(h(\mathbf{x}) - y)^2]$. Show that among *all* functions, the one which minimizes E_{out} is $f(\mathbf{x}) = \mathbb{E}_{P(\mathbf{x},y)}[y|\mathbf{x}]$.

(b) To estimate f, first estimate $P(\mathbf{x}, y)$ and then compute $\mathbb{E}[y|\mathbf{x}]$. Treat the N data points as N "unsupervised" points $\mathbf{z}_1, \ldots, \mathbf{z}_N$ in $\mathbb{R}^{d+1}$, where $\mathbf{z}_n^{\mathrm{T}} = [\mathbf{x}_n^{\mathrm{T}}, y_n]$. Suppose that you use a GMM to estimate $P(\mathbf{z})$, so the parameters $w_k, \boldsymbol{\mu}_k, \Sigma_k$ have been estimated ($w_k \geq 0$, $\sum_k w_k = 1$), and

$$P(\mathbf{z}) = \sum_{k=1}^{K} \frac{w_k |\mathrm{S}_k|^{1/2}}{(2\pi)^{(d+1)/2}} e^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu}_k)^{\mathrm{T}}\mathrm{S}_k(\mathbf{z}-\boldsymbol{\mu}_k)},$$

where $\mathrm{S}_k = \Sigma_k^{-1}$. Let $\mathrm{S}_k = \begin{bmatrix} \mathrm{A}_k & \mathbf{b}_k \\ \mathbf{b}_k^{\mathrm{T}} & c_k \end{bmatrix}$ and $\boldsymbol{\mu}_k = \begin{bmatrix} \boldsymbol{\alpha}_k \\ \beta_k \end{bmatrix}$. Show that

$$g(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \frac{\sum\limits_{k=1}^{K} \hat{w}_k e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\alpha}_k)^{\mathrm{T}}\Omega_k(\mathbf{x}-\boldsymbol{\alpha}_k)} (\beta_k + \frac{1}{c_k}\mathbf{b}_k^{\mathrm{T}}(\mathbf{x} - \boldsymbol{\alpha}_k))}{\sum\limits_{k=1}^{K} \hat{w}_k e^{\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\alpha}_k)^{\mathrm{T}}\Omega_k(\mathbf{x}-\boldsymbol{\alpha}_k)\right)}},$$

where $\Omega_k = \mathrm{A}_k - \mathbf{b}_k\mathbf{b}_k^{\mathrm{T}}/c_k$ and $\hat{w}_k = w_k\sqrt{|\mathrm{S}_k|/c_k}$. Interpret this functional form in terms of Radial Basis Functions.

(c) If you non-parametric Parzen windows with spherical Gaussian kernel to estimate $P(\mathbf{x}, y)$, show that $g(\mathbf{x})$ is an RBF,

$$g(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \frac{\sum\limits_{n=1}^{N} e^{-\frac{1}{2r^2}\|\mathbf{x}-\mathbf{x}_n\|^2} y_n}{\sum\limits_{n=1}^{N} e^{-\frac{1}{2r^2}\|\mathbf{x}-\mathbf{x}_n\|^2}}.$$

[Hint: This is a special case of the previous part: what are $K, w_k, \boldsymbol{\mu}_k, \mathrm{S}_k$?]